# CANADIAN THESES ON MICROFICHE

# THÈSES CANADIENNES SUR MICROFICHE

National Library of Canada
Collections Development Branch

Canadian Theses on
Microfiche Service

Ottawa, Canada
K1A 0N4

Bibliothèque nationale du Canada
Direction du développement des collections

Service des thèses canadiennes
sur microfiche

## NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

## AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE

Canada

NL 339 (r. 85/09)

National Library of Canada
Bibliothèque nationale du Canada

Ottawa, Canada
K1A 0N4

TC –

ISBN 0-315-21268-3

CANADIAN THESES ON MICROFICHE SERVICE – SERVICE DES THÈSES CANADIENNES SUR MICROFICHE

## PERMISION TO MICROFILM – AUTORISATION DE MICROFILMER

• Please print or type – Écrire en lettres moulées ou dactylographier

### AUTHOR – AUTEUR

Full Name of Author – Nom complet de l'auteur

C. P. GANGADHARAIAH

Date of Birth – Date de naissance
16 January 1953

Canadian Citizen – Citoyen canadien
☐ Yes / Oui ☒ No / Non

Country of Birth – Lieu de naissance
INDIA

Permanent Address – Résidence fixe
#49, I CROSS (Besides P.O)
SUDHAMANAGAR
BANGALORE-560 027. INDIA

### THESIS – THÈSE

Title of Thesis – Titre de la thèse

DATA STRUCTURES FOR SPATIAL DATA PROCESSING

Degree for which thesis was presented
Grade pour lequel cette thèse fut présentée
M.Sc

Year this degree conferred
Année d'obtention de ce grade
1985

University – Université
ALBERTA

Name of Supervisor – Nom du directeur de thèse
WAYNE A. DAVIS

### AUTHORIZATION – AUTORISATION

Permission is hereby granted to the NATIONAL LIBRARY OF CANADA to **microfilm** this thesis and to **lend** or **sell copies of the film.**

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

L'autorisation est, par la présente, accordée à la BIBLIOTHÈQUE NATIONALE DU CANADA de **microfilmer** cette thèse et de **prêter** ou de **vendre des exemplaires du film.**

L'auteur se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans l'autorisation écrite de l'auteur.

ATTACH FORM TO THESIS – VEUILLEZ JOINDRE CE FORMULAIRE À LA THÈSE

Signature
C. P. Gangadharaiah

Date
12 Feb 1985

NL-91 (r. 84/03)

Canada

THE UNIVERSITY OF ALBERTA

# DATA STRUCTURES FOR SPATIAL DATA PROCESSING

by

C. P. GANGADHARAIAH

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE

OF Master of Science

Department of Computing Science

Edmonton, Alberta

Spring, 1985

THE UNIVERSITY OF ALBERTA

**RELEASE FORM**

NAME OF AUTHOR C. P. GANGADHARAIAH

TITLE OF THESIS DATA STRUCTURES FOR SPATIAL DATA PROCESSING

DEGREE FOR WHICH THESIS WAS PRESENTED Master of Science

YEAR THIS DEGREE GRANTED Spring, 1985

Permission is hereby granted to THE UNIVERSITY OF ALBERTA LIBRARY to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

(SIGNED) C. P. G.....

PERMANENT ADDRESS:

#49, I Cross

Sudhamanagar

Bangalore 560 027, INDIA

DATED 31st January....1985

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled DATA STRUCTURES FOR SPATIAL DATA PROCESSING submitted by C. P. GANGADHARAIAH in partial fulfilment of the requirements for the degree of Master of Science.

W. A. Davis

Supervisor

J. Calz

Masayoshi Ito

Ian Orme Tosh

D. L. W. Hall

Date 31st January 1985

## ABSTRACT

This thesis is concerned with the data structures for spatial data processing. A general hierarchical data format, namely the *binary tree* format for representing binary valued and gray scale images is presented. Algorithms for the image operations such as binary arrays to binary tree conversion, geometric operations such as neighbor finding, area computation, perimeter computation and set theoretic operations such as complement, union and intersection are presented. As an application of this data format, a method of building an n-dimensional image from a set of (n-1)-dimensional sectional images is presented. A data compression technique known as *linear binary trees* is given. Image operation such as encoding of black pixels, finding an adjacent pixel in any given direction, linear translation in any given direction, rotation by ±90° are developed for linear binary tree. Regular and linear binary tree data formats are evaluated and compared with other data formats for their space and time efficiencies.

## ACKNOWLEDGEMENTS

First, I would like to thank Dr. Wayne A. Davis for his supervision of this work, continued support and helpful comments at various instants.

I am grateful to the members of the examining committee consisting of Drs. S. Cabay, T. Ozsu, M. Aoki and J.C. Muller, for their critical review and helpful comments. Also, I would like to thank the Department of Computing Science for providing me with the financial support and the National Science and Engineering Council for supporting this work under the grant NSERC A7634.

Finally, I would like to thank my wife, Kala and the members of my family for the various form of help and support that they have given to me.

## Table of Contents


| Chapter | | Page |
|---|---|---|
| 1. | INTRODUCTION | 1 |
| 2. | SPATIAL DATA STRUCTURES: A SURVEY | 7 |
| | 2.1 TERMINOLOGY | 8 |
| | 2.2 NON-HIERARCHICAL STRUCTURES | 9 |
| | 2.2.1 Points | 9 |
| | 2.2.2 Polylines | 9 |
| | 2.2.3 Chain Coding | 11 |
| | 2.2.3.1 Generalized Chain Code | 13 |
| | 2.2.4 $\psi$-s Curves | 14 |
| | 2.2.5 Skeletons | 14 |
| | 2.2.6 Tightly Closed Boundaries | 16 |
| | 2.2.7 Boundary Networks | 19 |
| | 2.2.8 Binary Searchable Polygonal Representation | 20 |
| | 2.2.9 Irregular Tessellations | 22 |
| | 2.2.10 Image Data | 23 |
| | 2.2.10.1 List Structure | 24 |
| | 2.2.10.2 String Representation | 24 |
| | 2.2.10.3 Run-Length Encoding | 25 |
| | 2.2.11 Hybrid Structures | 26 |
| | 2.3 HIERARCHICAL STRUCTURES | 27 |
| | 2.3.1 One Dimension - Bit-trees | 27 |
| | 2.3.2 Strip Trees | 28 |
| | 2.3.3 Quadtrees | 31 |
| | 2.3.4 Oct-trees | 34 |
| | 2.3.5 Hex-trees | 36 |

2.3.6 Linear Quadtrees .....36
2.3.7 Point Quadtrees and Line Quadtrees .....37
2.3.8 Pyramids .....39
2.3.9 Triangular and Hexagonal Tessellations ...39
2.3.10 Peano-scans .....40
2.3.11 K-D Trees .....42
2.3.12 The RSE-Structure .....44
2.4 SUMMARY .....45
3. BINARY TREES .....47
3.1 INTRODUCTION .....47
3.1.1 Binary Trees .....49
3.2 SYMMETRIC BINARY TREES .....50
3.2.1 Properties of Symmetric Binary Trees .....51
3.2.2 Definitions .....53
3.2.3 Neighbor Finding Technique .....57
3.2.4 Computing Perimeter of Regions .....60
3.2.5 Binary Trees from Binary Arrays .....62
3.2.6 Area of the Image Represented by a Binary Tree .....64
3.2.7 Set Operations .....65
3.2.8 From Lower to Higher Dimensional Images ..69
3.3 ASYMMETRIC BINARY TREES .....79
3.3.1 Properties of Asymmetric Binary Trees ....79
3.4 LINEAR BINARY TREES .....82
3.4.1 Encoding of Black Pixels .....83
3.4.2 Adjacency .....86
3.4.3 Linear Translation .....88

3.4.4 Rotation .....90
3.4.5 Set-theoretic Operations .....92
4. EVALUATION .....93
4.1 SPACE EFFICIENCY .....94
4.1.1 Best Case .....94
4.1.2 Worst Case .....95
4.1.3 Comparison .....100
4.1.4 Time Efficiency .....103
5. CONCLUSION AND FUTURE WORK .....105
REFERENCES .....107

# CHAPTER 1

## INTRODUCTION

This thesis is concerned with the data structures for spatial data processing. Spatial data structures are receiving increasing attention from researchers in computer graphics, image processing, computer cartography and related fields, see [3, 20, 23, 24, 41]. There is an increasing amount and range of spatial data available in digital form and an increasing need for their use in cartography, medical image processing, etc. The versatility of spatial data processing systems cannot accommodate this broader range of applications, as well as incorporating different types of data from a variety of sources [32]. The lack of versatility is compounded by the fact that, spatial data volumes are ever increasing, hence there is a problem of physical storage and time needed for access and processing. These problems are attributed in a large part to the differences in commonly used storage formats.

The term *spatial* data applies to any data concerning phenomenon which is areally distributed in 2 or more dimensions. This includes such things as: Computed Tomographic (CT) scanned images representing sectional views of a part of the human body; geographic data, which may be two dimensional, modeling the surface of the earth as a

plane, or three dimensional, modeling surface features such as elevations and depressions. Spatial data might even include time as a fourth dimension, for the representation of a beating heart, inhaling and exhaling of lungs, animation sequences, etc.

Spatial data may be clasified into several types. The first is point data where each element is associated with a single location, such as the location of cities or towns on a large scale map. The second is line data. With this data type, line structures such as rivers, roads are represented. The third type is polygonal data, where the location of a data element is represented by a closed sequence of lines. Polygonal data is associated with areas over a specified space. This data can be any one of the three following categories, namely (1) isolated polygons, (2) adjacent polygons, and (3) nested polygons. A fourth category can be considered to consist of some mixture of the above types.

Spatial data models have a number of characteristics which significantly differentiate them from others. For example, geographic data is very complex because of natural phenomenon which is very irregular and complex. A combination of properties such as multi-dimensionality, and the complex nature makes modeling geographic data uniquely difficult.

Data structures are basically a detailed representation of abstract data namely a data model. As the process of data model design is essentially one of abstraction, no model or

abstraction of reality can represent all aspects of it. Hence, it is impossible to design a general purpose data structure that would be equally efficient in all situations. Hence some spatial data structures are good for plotting, but may be very inefficient for analytic purposes and vice versa.

As data structures are built upon data models, and detail the data elements, they are regraded as structural models too. Data structures are usually organized into lists, arrays, etc. The relationships between the objects, or data elements may be expressed explicitly or implicitly. Explicit relations are written into the data structure as data elements themselves. Implicit relationships can be indicated by the relative positions of individual data elements and some of the relations can be computed.

An effective spatial data structure needs to posses the following qualities:

1. Efficient (in both time and space).
2. Easy encoding.
3. Easy manipulation.
4. Data compression should be possible.
5. Easily convertible to output formats.
6. Robust,
7. Flexible to accommodate changes.
8. Easy addressing.
9. Able to handle data of different dimensions.

There are several very important advantages of a regular, recursive tessellation of planes as a data model. This gives rise to a hierarchical model. As a result, this particular type of data model is currently receiving a great deal of attention. The recursive subdivision of space facilitates physically distributed storage and helps in operations such as browsing, windowing, etc.

Most popular hierarchical structures used in the spatial data processing applications are quadtrees for 2-D images and oct-trees for 3-D images. These data structures are well studied and algorithms for most of the geometric operations such as neighbor finding, area computation, perimeter computation and set theoretic operations such as complement, union and intersections are well established. Additionally, there are data compression techniques such as those proposed by Gargantini [16], Jones and Iyengar [22], and Davis and Wang [7], on these structures. These techniques provide a substantial savings in the memory requirement at the cost of processing time. In the case of quadtrees and oct-trees the branching factor increases exponentially as the dimensionality of the space increases ($2^n$, for an n-dimensional space). Hence the tree structure is different for images of different dimension. For example, in a quadtree all non-terminal nodes will have four sons and in an oct-tree all non-terminal nodes will have eight sons. Another approach is to keep the branching factor the same and incorporate the dimensionality implicitly in the

structure. This approach gives rise to binary trees which can be used for images of any dimension. The purpose of this thesis is to study this other approach and analytically establish how good (or bad) this data format is compared to other data formats. It is shown that a binary tree data format can be used for images of 2 or more dimensions. Since the tree structures are independent of the dimensionality of the space, the developed algorithms can be used for images of any dimension.

Chapter II of this thesis deals with the terminology used and a survey of hierarchical and non-hierarchical data formats used in the representation of spatial data.

Chapter III deals with a binary tree data format. Algorithms are developed for neighbor finding in any given direction, raster to binary tree conversion, area of the image, set theoretic operations such as complement, union and intersections of two images. As an application of this data format, an algorithm is developed to build an n-dimensional space from a set of (n-1)-dimensional sectional images. This is of potential use in the medical image processing. For example, Computed Tomography (CT) provides a set of 2-D image sections of a part of a human body. From this set, a 3-D image can be built from the given algorithm.

A data compression technique is discussed and the resultant tree is called a *linear binary tree*. The discussion includes algorithms for: encoding of black

pixels, finding an adjacent pixel in any given direction, linear translation and rotation by ±90°.

Chapter IV is concerned with the space and time efficiency of regular and linear binary trees. It is shown that in the worst case binary trees require 12.5% more nodes than quadtrees for the representation of 2-D images, but linear binary trees require 25% less space than linear quadtrees. The binary tree data format is compared with other data formats for space and time efficiency.

Chapter V presents the conclusions and suggestions for future work.

# CHAPTER 2

## SPATIAL DATA STRUCTURES: A SURVEY

This Chapter deals with the terminology, definitions and an overview of spatial data structures.

There are several primitives conventionally used in spatial data. The first is point data, where each data element is associated with a single location in space, such as the location of cities or towns in a large scale map. The second is line data. With this data type, the location is described by a string of spatial coordinates. These are used for representing such things as: roads, rivers, etc. The third type is region or polygonal data, where the location of a data element is represented by a closed string of spatial coordinates. The fourth data type is an image, which is some mixture of the above types. The division between these four primitives is somewhat difficult, because a set of points give rise to lines and through a set of lines one can represent regions and even images. Never-the-less the physical and logical organization of these data elements is always relevant in the analysis. The data formats used in the representation of the primitives can also be broadly classified into two categories: namely *hierarchical structures* and *non-hierarchical structures*. In the hierarchical structures, a whole picture is broken into

parts, the resulting parts into smaller parts, and so on, down to the level of primitives in the line drawing or pixels in gray tone images. The data structure used to represent hierarchic information is a tree. Other methods of representing the picture data falls into the category of non-hierarchical structures.

## 2.1 TERMINOLOGY

A 2-D *digital image* is a mapping F(p) that associates each pixel p with a value. When the 2-D image is a continuous function, it is sampled and quantized. This process represents the pixel value with finite precision. Usually the pixels are represented as integers (gray levels or colours) in a given range D, where D is a set of integers.

Without loss of generality, a 2-D *binary image* is defined by the function f(p), whose domain is the set of all pixels p and the range D is the set {0,1} or {white, black}. The set of pixels S = {p|f(p) = 1} is referred to as the image and the set S' = {p|f(p) = 0} as the background.

In general, given nonzero positive integers m and n, H(m,n) is a hypercube centered at the coordinate axis $I_0, I_1, \ldots, I_{m-1}$, where m is referred to as the dimensionality of the space and n as the resolution of digitization.

## 2.2 NON-HIERARCHICAL STRUCTURES

This section deals with non-hierarchical data formats used in the representation of primitives.

### 2.2.1 Points

Information regarding points can be represented in either absolute or relative coordinates. A straightforward representation of a set of points is a simple array. In this representation each subscripted variable in a predefined array represents a point, which is uniquely determined by its coordinates. A point in k-dimensions can be identified by storing the absolute coordinate of the point in the array. To represent a set of points in k-dimensions, each element in the array will have k entries.

Points can also be represented by relative coordinates. In this case each point is represented by the difference in the coordinates with respect to the previous point, or to some specified reference point.

### 2.2.2 Polylines

*Polylines* or *Polygonal lines* are a straightforward method of representing line information. In this representation, the point list $k_1$, $k_2$, ... , $k_n$ represents the concatenation of the line segments from $k_1$ to $k_2$ and from $k_2$ to $k_3$ and so on. If the first point is same as the last, a closed boundary is represented.

Polylines can approximate curves to any resolution of accuracy. There are many different approaches possible for approximating a given curve. Finding a satisfactory approximation to a given curve basically involves segmentation. The problem is to find corners or break points that yield the best polyline.

A simple method of generating polylines is known as merging [4]. The method is as follows: points along a curve are considered in order and approximated to a linear segment as long as they fit within some tolerance. When they do not, a new segment is begun.

Another method of polyline generation is known as the splitting scheme [8]. In this scheme, line segments are divided when they fail some fitting condition. The following algorithm provides an example.

**Algorithm for Curve Approximation**

1. Given a curve as in Fig. 2.1a, draw a straight line between its end points (Fig. 2.1b); call it the approximating line.
2. For every point on the curve, compute its perpendicular distance to the approximating line. If the perpendicular distance from all points is within some tolerance, exit.
3. Otherwise, pick the curve point farthest from the approximating line, make it a new break point (see Fig. 2.1c) and replace the relevant segment of the polyline with two new line segments.

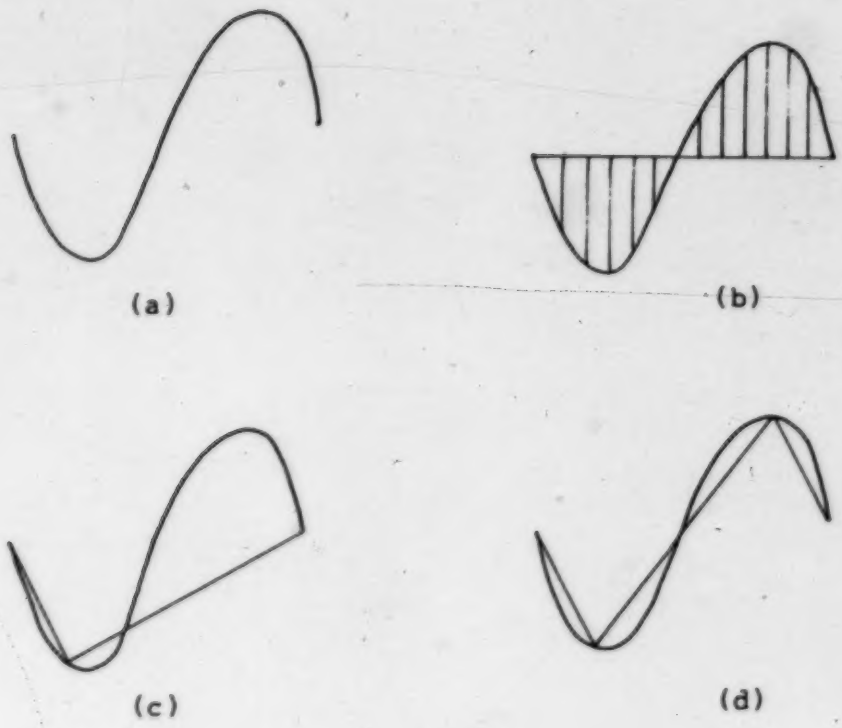



Fig. 2.1. Stages in curve approximation.

4. Recursively apply the above algorithm to each of the two new segments (see Fig. 2.1d).

### 2.2.3 Chain Coding

Information regarding lines can also be represented by chain coding. Chain codes are also known as Freeman-Hoffman codes [10]. Chain coding consists of assigning a unique directional code between 0 and 7 for each of eight unit length vectors as shown in Fig. 2.2. A line structure can

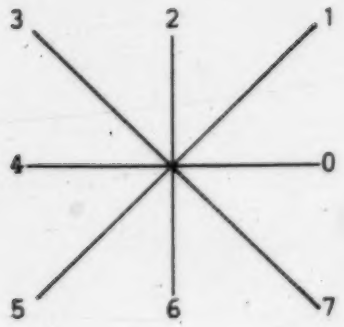



Fig. 2.2. Chain code assignment for an 8 point chain code.

therefore be represented by a sequence of octal digits. Each octal digit corresponds to one directed straight line segment. These straight line segments are referred to as links and a sequence of links representing a line structure is known as a chain.

*Definition 1:* A link $a_i$ is a directed straight line of length $T(\sqrt{2})^n$ and of angle $a_i \times 45°$ referred to the x-axis, where $a_i$ is any integer from 0 to 7, n is the modulo-2 value of $a_i$ and T is the grid size.

*Definition 2:* A chain is an ordered sequence of links with possible interspersed control codes.

$$A = a_1\, a_2 \ldots a_n$$

*Definition 3:* The links $a_i$ and $a_j$ form an inverse pair if $a_j = a_i + 4 = a_i^{-1}$ where + represents modulo 8-addition.

The octal digit sequence $04d_1d_2$ is reserved as a control code indication and does not denote a pair of links unless $d_1d_2 = 04$.

2.2.3.1 Generalized Chain Code

The natural extension of a chain code is to use straight line segments with an additional set of permissible lengths or orientations or both. In the basic (8-point) chain code, the next node in sequence for a given present node is one of the 8 nodes that are at a distance of 1 or $\sqrt{2}$. Instead of only 8 permissible nodes there could be a 24 point scheme where the permissible link length is 1, $\sqrt{2}$, 2, $\sqrt{5}$ and $2\sqrt{2}$. There also will be 16 allowed directions. A curve encoded with such a scheme is likely to exhibit finer angular quantization and to contain fewer segments. Based on the same principle a variety of other chain coding schemes can be readily postulated. This coding scheme is known as *Generalized Chain Coding* (GCC) [13].

GCC can be modified to take advantage of the inherent slope coherence between two successive chain links. Rather than encoding each new link directly, the new link end point is estimated using the previous link and the first backward difference based on the preceding link. The coding matrix is then used to encode the error between this estimated end node and the actual end node.

This scheme is known as *Generalized Chain Difference Coding.* (GCDC) [15].

### 2.2.4 $\psi$-s Curves

$\psi$-s curves provide a basis for a measure of shape [2]. These curves are like a continuous chain code representation. $\psi$ is the angle between a fixed line and a tangent at the boundary of a region. It is plotted against the arc length (s) of the boundary traversed. For a closed curve, the function is periodic.

Horizontal lines in $\psi$ curves correspond to straight lines on the boundary. Non-horizontal lines correspond to segments of circles. Thus a $\psi$-s curve may be segmented into straight lines, yielding a segmentation of the boundary in terms of straight lines and circular arcs (see Fig. 2.3).

### 2.2.5 Skeletons

A method of representing an arbitrary planar region is called a *skeleton* or *medial axis transform* [33, 34]. This data format provides a compact representation for images. The concept is based on the principle that any given region can be described as a union of *maximal neighborhoods* of the *skeleton* set of its interior points.

A digitized image is given in the form of a rectangular matrix of elements $a_{i,j}$ where i and j are the cartesian coordinates of the point $a_{i,j}$. In order to define the concept of maximal neighborhood, one should specify a metric of the

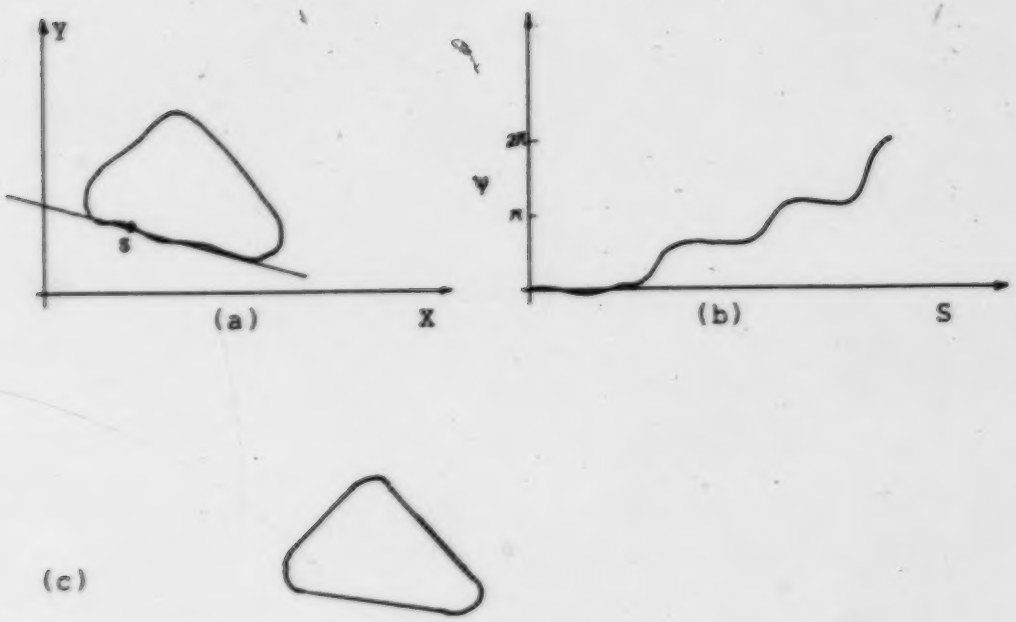



Fig. 2.3. Segmentation. (a) Curve and a tangent, (b) Curve showing a region of high curvature, (c) Resultant segment.

picture matrix. Let $P_1(i_1, j_1)$ and $P_2(i_2, j_2)$ be two points and define $d(P_1, P_2) = |i_1-i_2|+|j_1-j_2|$. This function has the following properties:

1. $d(P_1, P_2) \geq 0$ and $= 0$ iff $P_1=P_2$,
2. $d(P_1, P_2) = d(P_2, P_1)$,
3. $d(P_1, P_3) \leq d(P_1, P_2) + d(P_2, P_3)$,

for all points $P_1$, $P_2$ and $P_3$.

Consider any point $P_0(i_0, j_0)$. The neighborhood of $P_0$ of radius r, where r is any non-negative integer, is defined as the set of all P(i, j) such that $d(P, P_0) \leq r$. This neighborhood is just the square array of points centered at

$P_o$, oriented diagonally with side r+1 points long, as shown in Fig. 2.4. If r=0, then the neighborhood reduces to $P_o$ itself.

If R is a open region, which is a subset of a given region M. Let $P_o$ be any point in R. Some neighborhood of $P_o$ must always be contained in R. Let $N_i$ be the set of all neighborhoods of points of R, such that each neighborhood is contained in R. Their union must be all of the region R. A neighborhood in $N_i$ will be called maximal if it is not properly contained in any other such neighborhood. Since $N_i$ is finite set, any $I \in N_i$ is contained in at least one maximal neighborhood. Let $M_i \in N_i$ be set of all maximal neighborhoods, then R = union i, for $i \in M_i$.

Any neighborhood is defined by its centre and radius. Since any region is a union of maximal neighborhoods, it can be completely described by giving the centre and radius of each of these maximal neighborhoods. This region representation is known as the skeleton or medial axis transform, because the set $M_i$ may be visualized as a skeleton or median axis of the region. This representation is particularly suitable for certain region queries such as finding whether a given point is inside the region or not. Also set theoretic operations can be performed efficiently.

### 2.2.6 Tightly Closed Boundaries

This data structure [27] is used for the representation of line structures, contour maps and region coverage. It is

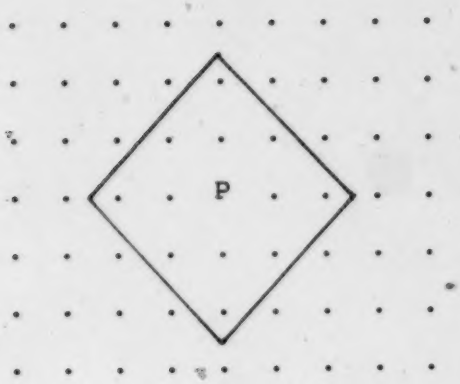



Fig. 2.4. Neighborhood of point P with radius 2.

efficient for searching large data files associated with geometric positions.

Consider the boundary of a region. The region is defined by the locus of a point in a rectangular grid. The locus is structured such that it is very easy to determine if a point (x, y) is within the region. A well known technique to do this is to draw a line from a point outside the region boundary to (x, y), then (x, y) is in the region if and only if the boundary is crossed an odd number of times by the line. This property holds provided the following restrictions are met.

1. The boundary must be continuous and closed. The absolute distance between the successive points in the locus cannot be greater than the grid element diagonal. While scanning the boundary locus, if it is found that this restrictions does not hold, the missing coordinates are

joined and made continuous.

2. Provisions must be made for the test line intersecting the boundary tangentially. This is done by augmenting every local maximum and the minimum in the boundary so that a tangential horizontal line always passes trough an even number of boundary points at each extremum.
3. Closed boundary cannot contain any loops.

A closed boundary with augmented extrema as described above is called a *Tightly Closed Boundary* (TCB).

The data format of a TCB is organized as follows: The coordinates of the TCB are partitioned into sets so that each set contain only points which have the same Y-coordinates. The associated X-coordinate of each of the sets are ordered monotonically increasing. Each of these sets are known as the Y partition of the TCB. The region $R_k$ having TCB $P_k$ is defined as follows:

$P_k = \{y^k_{mn}, y^k_{mx}, y^k_1, \ldots, y^k_n\}$, where: $y^k_{mn}$ is the smallest y coordinate in the boundary, and $y^k_{mx}$ is the largest. The y partition of $P_k$ for $y_i$ is

$y^k_i = \{r_{ik}, x_1^{ik}, \ldots, x_n^{ik}\}$, where $r_{ik}$ is the number of $y_i$ coordinates in the TCB of $R_k$ and is always an even number. Apart from storing the above points, the extreme X values, i.e., $X_{mn}$ and $X_{mx}$ of the TCB are determined while the above preprocessing is performed. These X values along with the $Y_{mn}$ and $Y_{mx}$ define a rectangular area bounding the TCB that is known as the *Geometric Index* of the TCB.

Given a region A, defined by a TCB structured into Y-partitions, it is possible to answer questions such as:

1. Is a given point within or on the boundary of A?
2. What is the area of A?
3. What is the minimum distance between a given point and the boundary of A?
4. What is the union of two given regions?
5. What is the intersection of two given regions?

### 2.2.7 Boundary Networks

Boundary networks are [26] a computer representation of the boundary of a region. The representation scheme is to efficiently determine the total or partial inclusion of arbitrarily given points and lines with respect to a set of general polygonal domains which partition a plane bounded region. The method also employs several levels of selection criteria for the purpose of reducing the number of accesses to auxiliary storage devices.

In this representation the bounded region is partitioned into a set of domains $\{D_i\}$, where $i=1,2,\ldots,N$. The partitioning is specified by means of boundaries consisting of an ordered sequence of connected straight line segments.

**Some Definitions**

1. *A boundary segment* is any one of the collection of line segments which are used to specify the partitioning

boundaries.

2. *A boundary node* is an end point which is common to three or more boundary segments.
3. *An ordered sequence of boundary segments* are the enumeration of boundary segments such that each segment is connected to its predecessor and successor at opposite end points.
4. *Significant points* are the end points which are local extremes in either X or Y coordinate values.
5. *A boundary section* is an ordered sequence of boundary segments beginning and ending at significant points and containing no intermediate significant point.

For the representation of the boundary of a region, the boundary sections are chosen as the basic elements to be processed. Since the boundary section is completely determined by the ordered sequence of boundary end points, these points are stored as the single retrievable entry in the storage media. In addition to the end points, the domains $\{D_i, D_j\}$ separated in whole or in part by the section are also stored. Finally each section is assigned a unique identification label such that the location of the section can be easily determined.

### 2.2.8 Binary Searchable Polygonal Representation

This is a representation scheme for polygons and polygonal lines which allows sets of consecutive sides to be collectively examined. The representation is known as the

*Binary Searchable Polygonal Representation* (BSPR) [6].

A polygon will be considered as a polygonal line with identical end points. The set of consecutive sides are known as sections. A set with only a single side is known as a primitive section. A section which is monotonic in both X and Y coordinates is known as a simple section. A significant point of a polygonal line is a vertex where there is an extreme in either X or Y coordinate values or both. The minimum and maximum X and Y coordinate values of the vertices of the section determine the smallest rectangle with sides parallel to the axis containing the section. This rectangle is known as the section rectangle.

Suppose a polygonal line has n sides (n+1 vertices) and m basic sections (m+1 significant points), then the BSPR consists of an ordered list of pointers to significant point (m+1 words) followed by an ordered list of minimum and maximum X and Y coordinates. These coordinate values correspond to the first m elements of C, where C denotes the set of all compound sections, followed by the ordered list of all vertices. Pointers to the significant points make it possible to locate the endpoints of elements of C, as well as the end points of basic sections.

BSPR's usefulness has been demonstrated [6] by two algorithms to find a point in polygon and intersection points. BSPR facilitates the rapid processing of polygonal lines for these operations.

### 2.2.9 Irregular Tessellations

Three most commonly used irregular tessellations are square, triangular and thiessen polygon meshes. The advantage of an irregular tessellation is that the structure of the mesh itself can be tailored to the areal distribution of the data, hence redundant data can be eliminated. An irregular mesh can be adjusted to the density of the data occurrences, hence cells become larger where the data is sparse and smaller where the data is dense.

Irregular tessellations are most frequently used as spatial data models in representing triangulated irregular networks (TIN), where each vertex of the triangulated mesh contains an elevation value. TIN's are a standard method of representing terrain data, and are mostly used for hill shading and hydrological applications.

Thiessen polygons, also known as Voronoi diagrams are constructed by bisecting the side of each triangle at a 90° angle (see Fig. 2.5), the result is an irregular polygon mesh where the polygons are convex and have variable number of sides. Thiessen polygons are useful for efficiently solving problems such as: the closest point, the smallest enclosing circle and others [42].

Irregular polygonal tessellations are uniquely suited to a particular type of data and particular analytical procedures. They are unsuited for most other spatial manipulations and analytical tasks. For example, overlaying two irregular meshes is extremely difficult. Generally

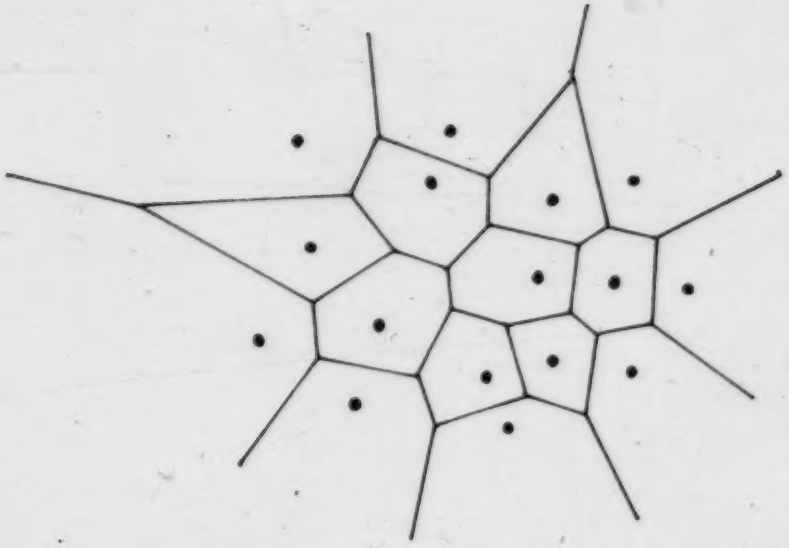


Fig. 2.5. The Thiessen Diagram.

irregular tessellations are also complex and time consuming. All these factors make irregular tessellations unpopular among the spatial data models, except in a very few specialized applications.

### 2.2.10 Image Data

This section deals with the data formats used in raster displays. The basic logical unit in a raster format is a scan line. A universal characteristic of the raster format is that the order of the data in the digital file is determined by its location in space and vice versa. Unlike vector data formats, spatial relationships are implicit in the raster data structure itself.

2.2.10.1 List Structure

A simple type of raster data organization is the square grid or matrix, wherein the individual pixels are stored as a sequence of zeros and ones indicating the absence or presence of an intersection of the grid and the contour. This is an uncompacted raster structure, where the basic logical unit is a single cell, or a pixel of the matrix. Each grid cell represents a location in two-dimensional space and the structure can be traversed in either the X or Y direction with equal ease. This structure stores the information regarding the cells. The address of a pixel is determined by the location of the cell in the sequential list. This type of data format is grid size dependent. If the grid size is larger, certain finer details of the contour may be left out.

Because of the simple nature of data organization, calculation of the perimeter of the polygon, or area enclosed by it is relatively simple. Windowing and clipping algorithms involve the calculation of the locations corresponding to the diagonally opposite corners of the rectangular window and selecting the pixels in between the two locations.

2.2.10.2 String Representation

As previously indicated, list structures have a uniform grid size. When the image is scanned horizontally it is possible to vary the grid size, also

it is required to eliminate the storage of redundant data values. Suppose the image consists of a number of objects, say $O_1$, $O_2$,...,$O_n$. In the horizontal scan of the image, if the length of the Kth object is $d_k$, then for each horizontal scan the string can be represented as $\{O_1d_1,\ldots, O_id_i ,\ldots, O_nd_n \}$ where the open and closed brackets indicate the starting and ending of a scan, and n is the number of objects found in a particular scan. While the image is being scanned, there could be certain horizontal lines where there may not be any object resulting in a blank scan. Such blank scans may be omitted. This is done by specifying the vertical dimension of the ith scan, say $v_i$ [11]. Thus the string representation looks like:

$$v_i(\ldots O_i\ d_i\ldots),$$

$$\ldots$$

$$v_j(\ldots O_j\ d_j\ldots).$$

This makes traversal of the data structure in the Y direction more difficult than in the X direction, since the exact position of a given X value within a scan line record will vary from one scan line to the next. The representation, however, gives a compact encoding of pictures with rectangular features [11].

2.2.10.3 Run-Length Encoding

This is a scan line model with compact data storage for images involving solid areas of gray tone or color. In a typical image, consecutive pixels will often have

the same intensity. Instead of storing the each intensity value separately, the length and intensity of each run of identical pixels is stored. Thus each encoded scan-line consists of one or more instructions, each instruction defining a run length and intensity value. The use of run-length encoding gives considerable savings in the amount of memory needed to store certain kinds of images [29].

**2.2.11 Hybrid Structures**

The data formats discussed so far fall into the category of either raster or vector. The basic logical unit of data in vector format is a point representing the start or a end of a vector. In this format spatial interrelations are either stored explicitly, thereby increasing the volume of stored data, or computed each time they are required. In the case of a raster format, the basic logical unit is a scan line. The spatial interrelationships are implicit in the structure itself. Each of these two types of spatial data structures offers significant storage versus processing tradeoffs.

A new data format proposed by D.J.Peuquet [31] is known as the *Vaster data format*. This format makes use of both raster and vector representations to utilize the advantages of both the formats. The basic logical unit of the vaster structure is a *swath*. Each swath spans a known range over Y which is a constant, and would correspond to a group of

contiguous scan lines if the data were organized in raster format. Each swath contains both a raster component and a vector component. The components are also recorded at the same grid resolution. The leading edge of each swath is recorded in raster format as a single scan line and functions as the index record for the swath, containing an identifier and X coordinate for each map line intercept. The raster encoding scheme contains all map line intersections within the same record. The data contained in the remainder of the swath are recorded in a vector format. All vectors contained in each swath are arranged in scan line intercept sequence from left to right. Each line intercept noted in the index record functions as the end point of each vector line segment within the swath.

## 2.3 HIERARCHICAL STRUCTURES

This section deals with hierarchical data structures used in the representation of spatial data.

### 2.3.1 One Dimension - Bit-trees

A bit-tree is a one dimensional binary tree which can be used for representing point data [19]. Consider a one dimensional picture (see Fig. 2.6), say the projection of a two dimensional picture onto an axis. A bit-tree may be constructed to represent this picture as follows:

1. Every node in the bit-tree of a picture of length $2^n$ represents a region of the picture whose size is $2^{(n-d)}$

units, where d is the depth of the node.

2. Every node n in the bit-tree is either a leaf or has exactly two sons.
3. Every node n in the bit-tree has a colour associated with it.

Note that no nodes will have two sons that are leaves and both colored the same, since this is more compactly represented by coloring the father node and removing the sons from the tree. See Fig. 2.6 for a linear picture and the corresponding bit-tree. A more comprehensive study of binary trees is presented in Chapter 3 of this thesis.

### 2.3.2 Strip Trees

This is a hierarchical representation scheme used for representing lines. It is a binary tree structure and each node of the tree consists of a special datum known as a *strip* and the tree is known as a *strip tree*. Lower levels in the tree correspond to the finer resolution of the line structure. A Strip tree data structure uses a special method of digitizing lines and retains all the intermediate information in the hierarchy [3, 30].

A strip S is defined as a six-tuple $(x_i, y_i, x_j, y_j, w_i, w_j)$. Where $p_i = (x_i, y_i)$ represents the beginning of the strip and $p_j = (x_j, y_j)$ represents the end of the strip. $w_i$ and $w_j$ are the right and left distances of the strip borders from the directed line segment $p_i$, $p_j$ (See Fig. 2.7). w is the width of the strip and is defined as $w_i + w_j$.

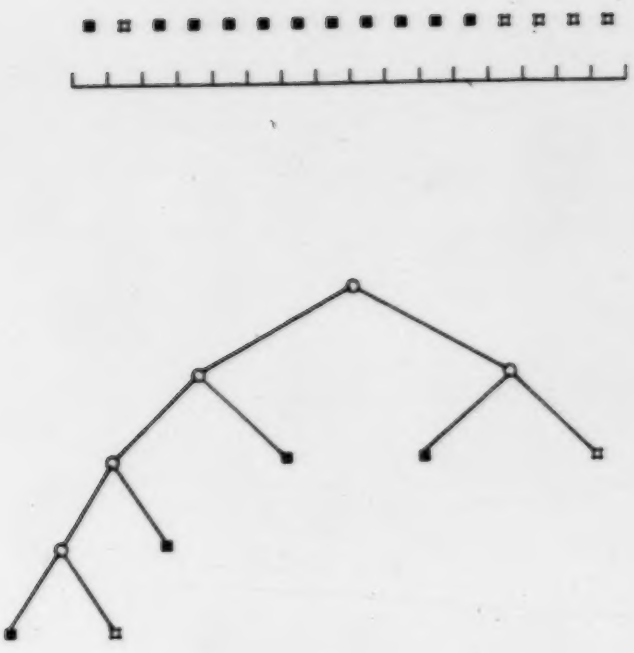


Fig. 2.6. A one dimensional picture and its bit-tree.

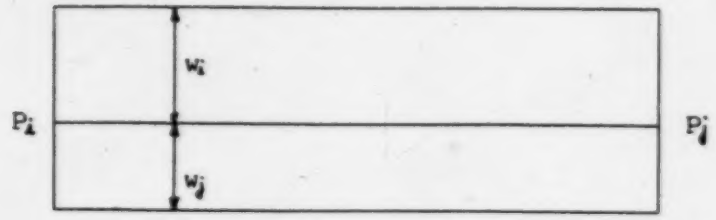



Fig. 2.7. Strip definition.

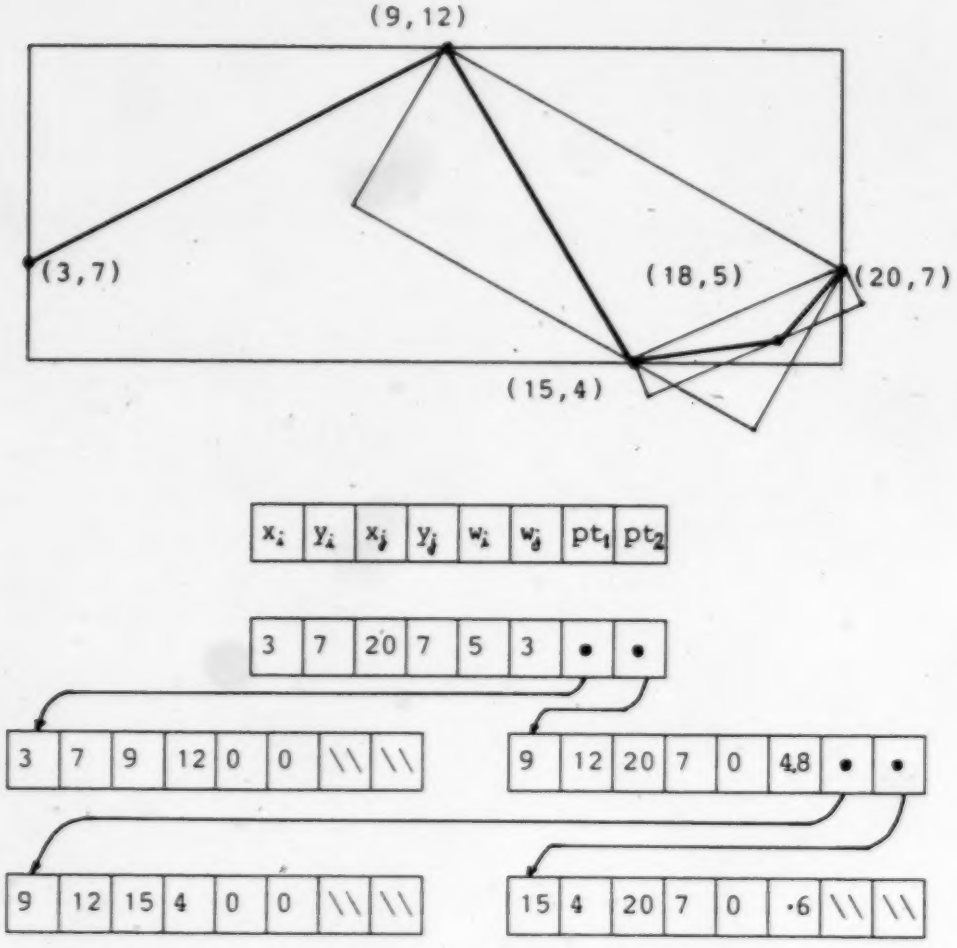



Fig. 2.8. Strip tree generation.

**Digitization**

A curve C is approximated by line segments with points $(x_0, x_1, \ldots, x_n)$. For any resolution, say $W^* \geq 0$, this line can be approximated with a strip as follows:

If L is a line segment connecting $p_0$ and $p_n$, find the smallest rectangle with sides parallel to L and which covers all the points. This rectangle is the strip of the root node of the strip tree. If $p_k$ is the point which touches one of the two sides of the rectangle that are parallel to L, then repeat the above process for each of the subsets $(p_0 \ldots p_k)$ and $(p_k \ldots p_n)$ This gives rise to two subtrees that are sons of the root node. This process terminates when all strips have $W \leq W^*$ (Fig. 2.8) where $W^*$ is the resolution of digitization.

The binary tree resulting from the above quantization process is known as a *Strip Tree*. The datum at each node is a strip S. A strip tree can be formally defined as either null or a node consisting of the six-tuple $(p_i, p_j, w_i, w_j,$ Lson, Rson) where Lson and Rson are strip trees that are either null or have root strips that are related to S by the digitization process.

**2.3.3 Quadtrees**

A top-down approach to regular decomposition of pictures of two dimensional arrays yields a tree structure. Such a structure gives successively deeper levels representing finer subdivisions of picture area. A quadtree

is one such tree representation of planar graphs, wherein a picture is partitioned into four sub-areas with the same shape and equal areas [20, 47]. The nodes of the tree are either leaves or have four children corresponding to each of the four areas. This technique of picture decomposition is used best when the picture is a square matrix whose side is a power of 2, say $2^n$. A quadtree can be formally defined as:

1. Every node in the quadtree represents a square area of the picture whose sides are of length $2^{(n-d)}$, where d is the depth of the tree.
2. Every node is a leaf, or has four sons each of which represents one quadrant of the area. The area is separated by dividing in half along each axis.
3. Every node has a color associated with it. If every pixel in the area represented is BLACK or WHITE, then the leaf is coloured BLACK or WHITE respectively. If the area associated with a node has a mixture of BLACK and WHITE colours, then the node is represented as GRAY and is a non-terminal node, hence will have four children, (see Fig. 2.9) [20, 47].

A method of organizing this data structure is to store each node as a record containing six fields. The first five fields contains pointers to the nodes FATHER and its four SONS which corresponds to four quadrants. The sixth field describes the contents of the block of the image which the node represents.

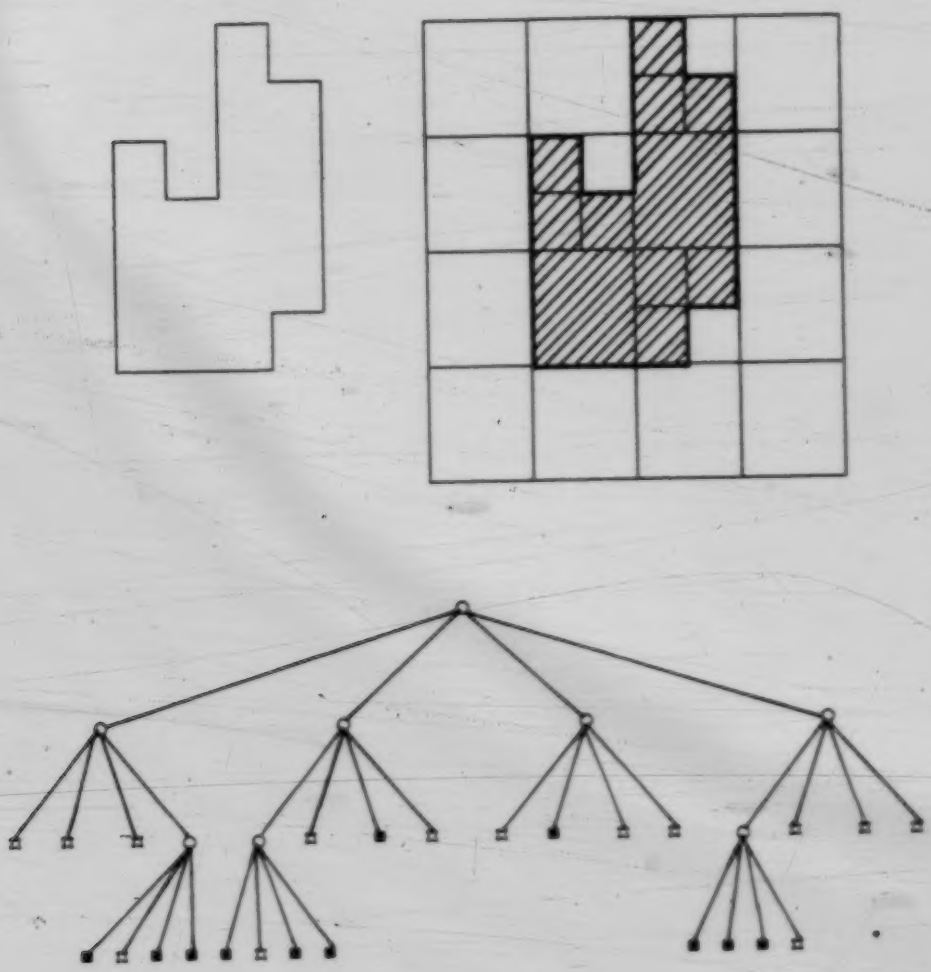


Fig. 2.9. Quadtree representation of a region.

### 2.3.4 Oct-trees

The extension of the quadtree principle to 3-D images yields an oct-tree data structure. Consider a 3 dimensional space of side $2^n$. At each stage of the division, the cube is sub-divided into eight equal octants. One face of the universe cube is arbitrarily chosen to be the front, and the four octants adjacent to this face are labeled Front. Similarly, the other four octants are labeled Back, and a unit cube is called a Voxel (for volume element). Fig. 2.10 shows the subdivision of a cube, the labeling of the eight octants and the corresponding oct-tree.

The definition of oct-tree to represent three dimensional scene is analogous to quadtrees. Assume that the voxels in the picture are either BLACK or WHITE. Each node of the oct-tree is either a leaf node or has eight sons each of which represents an octant of the father. A node is a terminal node if all the voxels in the cube represented are of the same colour. Otherwise the node represents a non-terminal node, and hence will have eight sons.

A method of organizing this data structure is to store each node as a record containing ten fields. The first nine fields contains pointers to the nodes FATHER and its eight SONS which corresponds to eight octants. The tenth field describes the contents of the block of the image which the node represents.

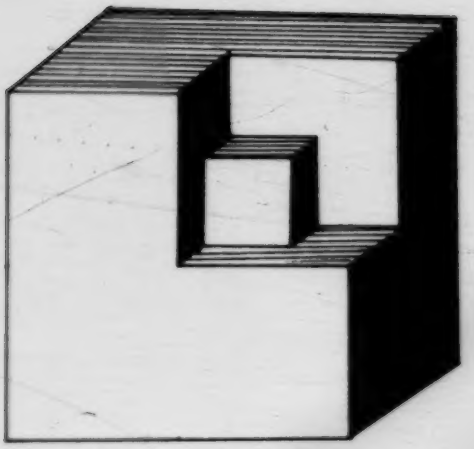


Fig. 2.10. Oct-tree representation of a region.

### 2.3.5 Hex-trees

An extension of same principle to a fourth dimension produces a hex-tree [19]. Since time is not measured in the same units as the other three dimensions, the extension is not as simple as the previous cases.

An image in the fourth dimension is usually conceived as various instances of a three dimensional image as it changes continuously through time. However for the purposes of storage, time is divided into a discrete number of steps. The basic unit of a hex-tree is called a *tixel* and consists of a unit of time and a voxel in 3-D space.

### 2.3.6 Linear Quadtrees

Quadtrees may be represented effectively without using pointers. This method is due to Gargantini [16]. In this method of representing quadtrees, only black nodes are encoded and are represented without using pointers with quarternary integers whose digits reflect the successive quadrant subdivisions. The sorted array of black nodes is referred to as a *linear quadtree*. This method of representing a quadtree has the property that:

1. Only black nodes are stored.
2. The encoding used for each node incorporates adjacency properties in the four principal directions.
3. The node representation implicitly encodes the path from the root to the node.

The main advantage of this structure is that the space and

time complexity depends only on the number of black nodes. Pointers are completely eliminated. Fig. 2.11 shows the quadrant labeling and generation of quarternary codes.

Extending the same concept to the oct-trees for 3-D images, produces linear oct-trees.

### 2.3.7 Point Quadtrees and Line Quadtrees

Point quadtrees are generated based on the location of ordered data points rather than regular spatial decomposition [12]. A point quadtree takes one data point as the root and divides the area based on this point. This process is repeated recursively for each ordered point in each quadrant which results in a tree of degree four. The relative location among the points yields a regular point data decomposition rather than a regular areal decomposition. Point quadtrees are useful in point search and nearest neighbor operation.

Line quadtrees are generated based on the location of a line in a grid format [40]. This format is used for hierarchically representing images which are segmented into a number of different regions. Line quadtrees encode both the region and its border in a hierarchical manner. This is in contrast to the conventional quadtrees which only encode areas in a hierarchical manner.

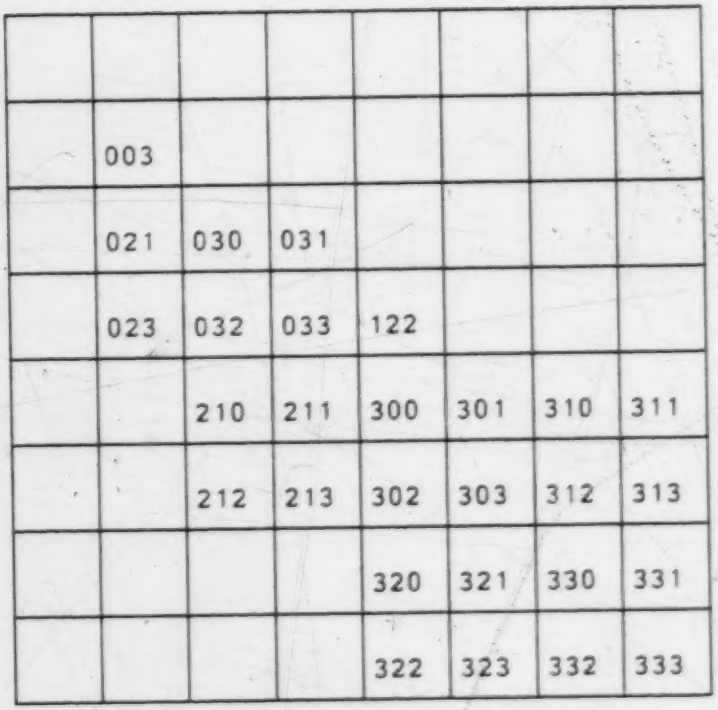



Fig. 2.11. Quadrant labeling for a linear quadtree.

### 2.3.8 Pyramids

The inverse process of forming quadtrees leads to the formation of *Pyramids* [44]. The formation of a pyramid starts from the base, which corresponds to the image at full resolution. Subsequent levels are obtained by combining 2x2 subregions in some manner. The operation is repeated until an array of managable size is obtained. The subsequent regions provide reduced resolution and hence reduce the computational complexity. Because of smaller data sets the problem of segmentation and feature extraction are simplified and also provides a convenient way of browsing the image data at a convenient level. Futhermore, processing can take place at the resolution required by the image.

### 2.3.9 Triangular and Hexagonal Tessellations

The three possible types of regular tessellations, namely: square, triangular and hexagonal meshes have been used as the basis of spatial data models. Each one has different functional characteristics which are based on the geometric properties of each of the models [1].

Triangular tessellations can also be recursively decomposed using triangular quadtrees. In a triangular quadtree each node corresponds to a triangle and thus has four children (Fig. 2.12). Each node will have three neighboring triangles, and one of the triangles will have one of the two orientations that differs by 60°. In comparison, a square quadtree has all squares in the same

orientation. Each square has four neighbors and exactly two of these are its siblings. In the case of triangular quadtrees, all neighbors of any node labeled 1 are its siblings. A node labeled 2, 3 or 4 on the other hand has exactly one sibling among its neighbors.

Hexagonal tessellations can be recursively decomposed as shown in Fig. 2.13. The primary advantage of hexagonal mesh is that all neighboring cells of a given cell are equidistant from that cell's center point. Radial symmetry makes this model advantageous for radial search and retrieval functions. This is unlike the square mesh where diagonal neighbors are not the same distance away as neighbors in the lateral directions.

### 2.3.10 Peano-scans

A method of transforming an n-dimensional space into a line and vice versa was discovered by Giuseppe Peano [32]. These are a family of curves which generate a track through space. These curves are known as peano-scans as-well-as space filling curves. These curves have the property of preserving spatial associativity of the scanned data space on a single dimension.

Peano-scans possess several properties, which can be useful in some spatial data handling applications. These are:

1. The curves pass through every locational element in the data space.

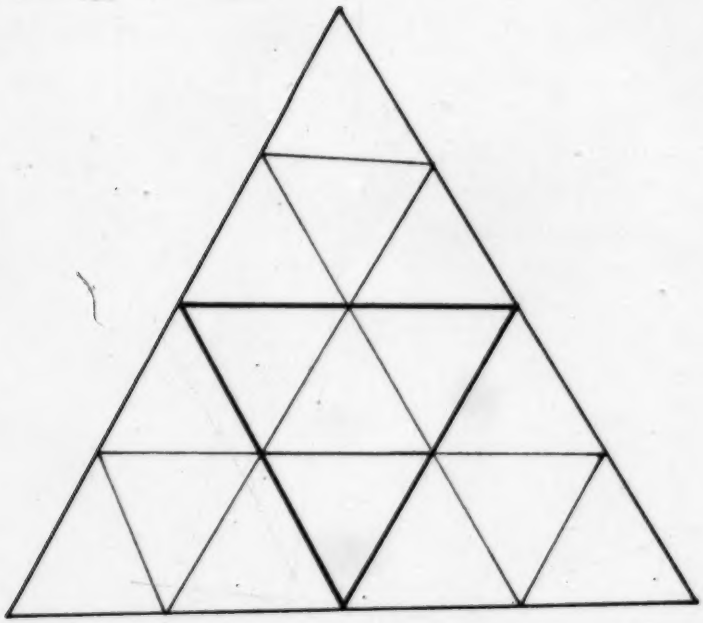


Fig. 2.12. Successive subdivision for a triangular quadtree.

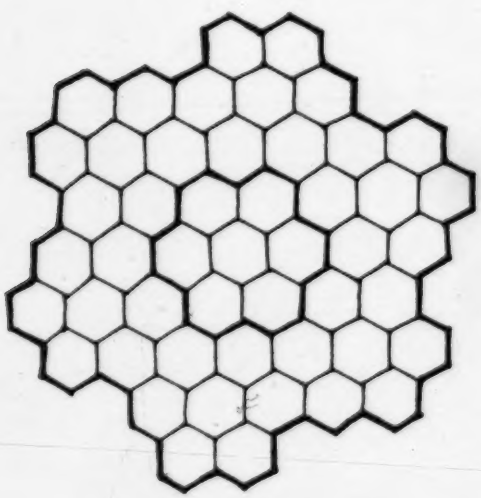


Fig. 2.13. Successive divisions of an hexagonal mesh.

2. The points close to each other in the curves are close to each other in space.
3. The curves act as transforms to and from themselves and the data space.

The practical application of peano curves is in the generation of areal indexing schemes. For indexing purposes, the spatial data base is usually divided into gridded areal data known as 'unit frames'. Each unit frame in the system is assigned a unique number starting at the origion of their coordinate system. The arrangement is shown in Fig. 2.14. The number scheme is known as a *Morton Matrix*. It is actually a Z-shaped peano scan.

**2.3.11 K-D Trees**

Multidimensional binary search trees (abbreviated K-D trees) was introduced by Bentley in 1975 [5]. This data structure is suitable for use in many multidimensional problems, specially for queries involving associative retrieval by secondary keys.

Consider a file F which is a collection of records R to be represented as a K-D tree. Each record R of F is an ordered K-tuple $(V_0,\ldots,V_{k-1})$ of values, which are keys or attributes of the record. In the K-D tree representation of the file, each node represents the record. Each node in addition to having K-keys has two pointers, which are either null (leaf nodes) or point to two other records in the tree. Associated with each node, is a discriminator which is an

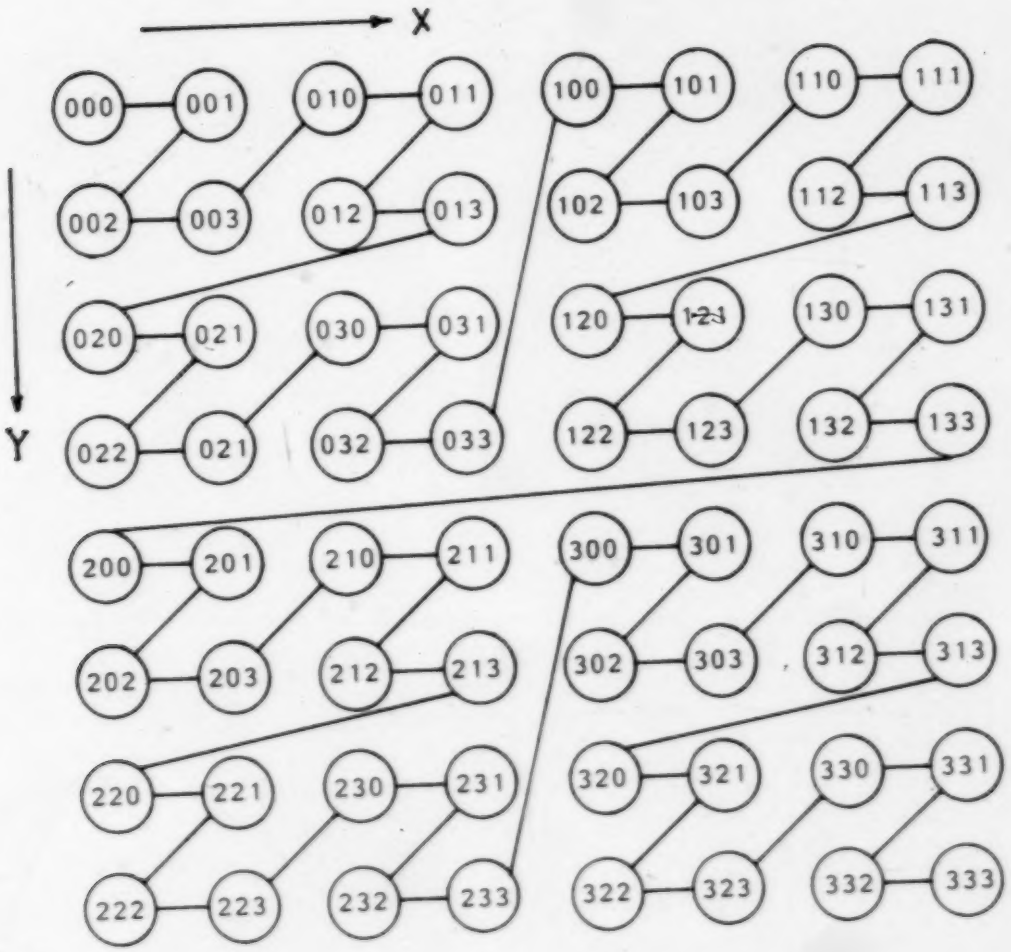



Fig. 2.14. Mortion Matrix Scheme.

integer from 0 to k-1. For any node p, the two pointers are defined as LOSON(p) and HISON(p) for the left and right sub-trees respectively. If j is the discriminator of p, then for any node Q in the LOSON(P), $V_j(Q) < V_j(P)$, and similarly for any node R on the HISON(P), $V_j(R) > V_j(P)$. All nodes at any given level have the same discriminator. The root node has the discriminator 0, its two sons have the discriminator 1 and so on to the K-th level in which the discriminator is k-1. The (k+1)st level has the discriminator 0 and the cycle repeats.

### 2.3.12 The RSE-Structure

A graph structure for the construction of a visual model is the RSE (region, segment, endpoints) structure [44]. The RSE structure is a representation of minimum information that must be contained in any symbolic representation of a segmented image. The RSE structure consists of a directed graph partitioned into three layers, namely: the *region plane*, which consists of nodes representing regions which are defined by the set of line segments, the *segment plane* which consists of nodes representing the line segments which form the boundary of the region, and *end point plane* which consists of endpoints of the line segments. There are sets of arcs from each of the region nodes to each of the segment nodes in the boundary of the region. There will be two arcs from each segment node to the two end points of the segments. There

will also be arcs from each end point node to the adjacent segments that the end point shares [44]. Fig. 2.15 illustrates the RSE structure.

## 2.4 SUMMARY

In summary, the two basic types, namely: hierarchical data structures and non-hierarchical data structures, have advantages and disadvantages which are inherent in the structure itself. Individual models, such as the ones discussed above, can overcome these problems only to a limited degree and always only by some sort of tradeoff. Vector models are a direct digital translation of map data, this means that the algorithms tend to be direct translations of traditional manual methods. Thus the vector mode algorithms are well developed and familiar. The primary drawback of vector models is that spatial relations must be either explicitly recorded or computed.

On the other hand, spatial relationships are built into regular tessellation type of data models. The drawback of this type of data model is that they tend to be not very compact. Regular tessellations force the storage of redundant data values. The redundancy of the data can be avoided by various compaction techniques. Sometimes the redundant data is useful in browsing and windowing.

It is clear that neither of the data models is intrinsically a better representation of space. The representation and algorithmic advantage of each one is data

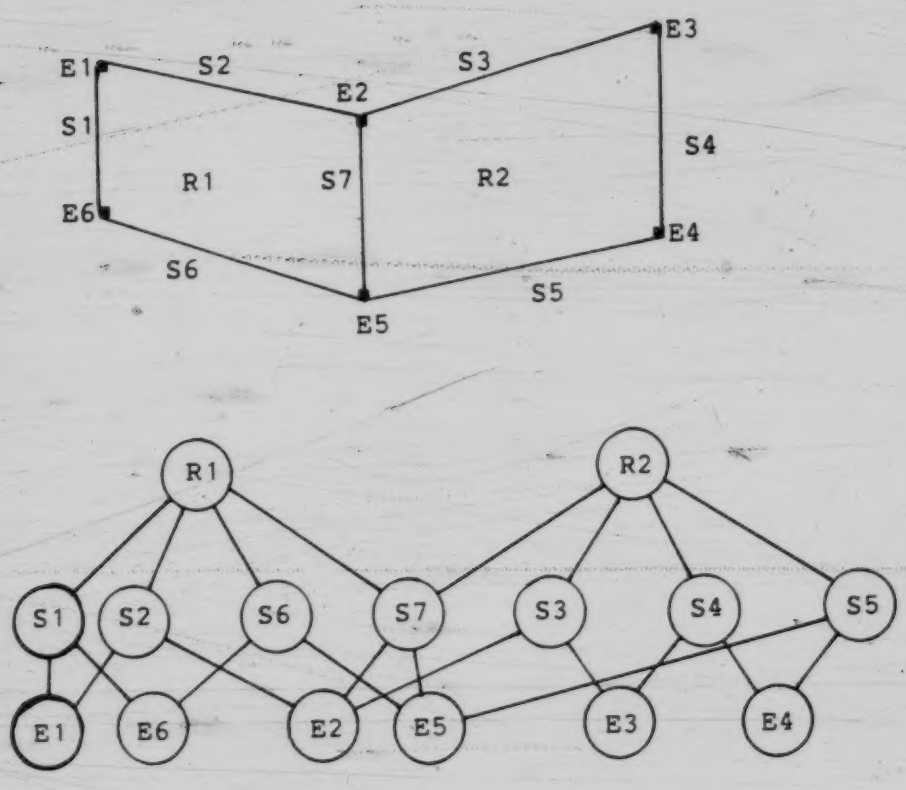



Fig. 2.15. The RSE-Structure.

and application dependent.

# CHAPTER 3

# BINARY TREES

The purpose of this chapter is to present a hierarchical data structure, which provides a general method for describing many algorithms, without regard to the eventual dimension at which they may be applied. An efficient method of encoding and storing a binary image using this data structure is discussed.

## 3.1 INTRODUCTION

Hierarchical data structures are receiving increasing attention from researchers in computer graphics, image processing, cartography and related fields [3, 20, 23, 24, 41]. Hierarchical representation of images is based on the principle of successive subdivision of the image. Since a digitized picture normally contains too many points for a meaningful description of the picture, it is required to partition these points in a systematic manner which enables easy processing. This involves the identification of a subset or a segment of the region. Hierarchical representations of pictures are an efficient way of picture segmentation which covers a large class of pictures and yet is easy to implement in terms of the computational complexity, storage requirements and extendibility.

Tree data structures lead to hierarchical representation of images. A 2-D image can be recursively subdivided into four quadrants until each quadrant is reduced to some simple form according to a given criterion. This process of picture segmentation is represented by quadtrees [10]. A three dimensional version of this method produces an oct-tree [20]. Most frequently, the division of the image area is motivated by issues such as image tessellations, ease of processing, general applications, etc. For example, an image which has triangular tessellations can be efficiently represented using triangular quadtrees [1]. There is considerable interest in quadtrees for 2-D images and oct-trees for 3-D images, see [4, 10, 12, 20, 35, 36, 41]. There are various compaction techniques suggested on these data structures. These include: linear quadtrees and linear oct-trees presented by Gargantini [16], virtual quadtrees, suggested by Jones and Iyenger [22], and coordinate based linear quadtrees, suggested by Davis and Wang [7]. These formats provide a space and time efficient representation of quadtrees and oct-trees. The generalization of recursive subdivision to four dimensions yields a hextree with 16 hexants [19]. Hextrees will be of potential use in the representation of time-varying three dimensional images. For example, dynamic spatial reconstruction of a beating heart, inhaling and exhaling of a lung, etc. While quadtrees, oct-trees and hextrees are constructed using the same principles,

structurally they are different, hence the algorithms to handle 2-D and 3-D images by these data structures are different.

The class of data structures presented in this chapter are called *binary trees*. This is a generalized version of a quadtree or an oct-tree. It can be applied in the representation of both 2-D images and 3-D images or, in general, it can be used for any m-dimensional spatial representation. Depending on how the region (for a 2-D image) is divided, either a *symmetric binary tree* or an *asymmetric binary tree* is produced. Properties of binary trees will be considered in more detail.

### 3.1.1 Binary Trees

Consider a 2-D binary image in an x-y coordinate system. A binary tree for this image is obtained by subdividing the image area into two halves by means of a line perpendicular to one of the coordinate axis. The whole image is represented as a root node and the two halves are represented as sons. At the next level of subdivision, a line is drawn perpendicular to the other axis and the process is continued until the area is reduced to a form which satisfies some given criterion.

**Definition 1:** A binary tree of an image can be defined as a finite set of nodes that is either empty or consists of a part of the image and at most two disjoint binary trees.

In the general case, to get a binary tree representing an m-dimensional space, the space is divided each time into two halves by means of a plane which is orthogonal to any one of the axes. Subsequent subdivisions are carried out by choosing the remaining axes in turn, until all the axes are chosen. This is done recursively until the m-dimensional space is reduced to a form which satisfies some given criterion. This results in a tree where each father node has at most two successor nodes.

Assume that each node is stored as a record. The fields in the record contain pointers to the node's father and its sons which corresponds to two sections. If P is a node and Q is a section, then these fields are referenced as FATHER(P) and SON(P,Q) respectively.

### 3.2 SYMMETRIC BINARY TREES

Consider a binary image corresponding to a $2^n x 2^n$ array of elements. A symmetric binary tree is formed by successive subdivision of the image area into an upper-half and a lower-half ($2^n x 2^{n-1}$) in the first level of the tree. In the next level the $2^n x 2^{n-1}$ areas are divided into $2^{n-1} x 2^{n-1}$ areas. In other words, in the formation of a symmetric binary tree, a given image area is divided into only two halves at each level of subdivision by means of an axis which is perpendicular to one of the axis and positioned midway along that axis. In the next level of the tree, the axis of division will be perpendicular to the previous axis

of division. The formation of a symmetric binary tree is shown in Figure 3.1.

In the general case, consider an m-dimensional space of side $2^n$ and coordinate axis $I_0, I_1, \ldots, I_{m-1}$. The symmetric binary tree of this space is obtained by first dividing the space into two halves by means of a plane which is orthogonal to the $I_{m-1}$ axis and positioned midway along that axis. The two halves (hyper-cubes) so obtained are referred to as $+I_{m-1}$ and $-I_{m-1}$ respectively. In the next level of subdivision, the axis $I_{m-2}$ is chosen, and each one of $+I_{m-1}$ and $-I_{m-1}$ spaces are further divided into two halves by means of a plane which is orthogonal to the $I_{m-2}$ axis and positioned midway along $I_{m-2}$ axis. The two halves so obtained are referred to as $+I_{m-2}$ and $-I_{m-2}$. This process is continued until all the axes are chosen and the whole process is repeated recursively until the space is reduced to a form which satisfies some given criterion.

### 3.2.1 Properties of Symmetric Binary Trees

Because symmetric binary trees of images are obtained by regular subdivision into only two parts, these trees have the following advantages:

1. In general, studying the case of binary division provides for a more general understanding of the structure and the results obtained can be applied to 2-D or 3-D images. Hence less attention is needed in

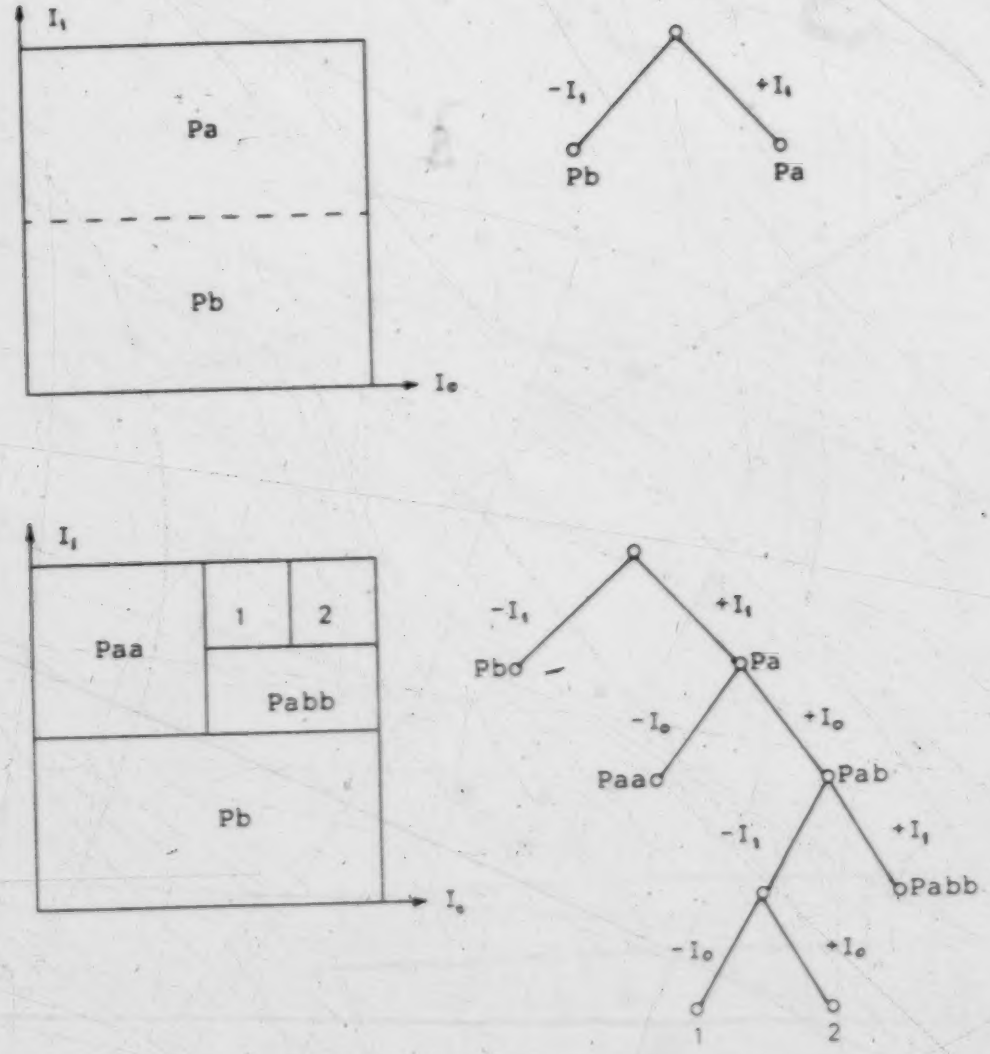



Fig. 3.1. A symmetric binary tree.

studying the results of different data structures.

2. Recursive subdivision of the space in this manner functionally results in a regular, balanced tree structure of degree two. Since the binary tree storage and search techniques is one of the more thoroughly understood topics of computer science, that knowledge can be easily applied.
3. Because the image is subdivided into a smaller number of parts as compared to quadtrees and oct-trees, the analysis of these trees should be easier.
4. All operations that can be done on quadtrees and oct-trees can also be done on binary trees.

### 3.2.2 Definitions

In the formation of the binary tree of a 2-D image, which is represented in the $I_0$ and $I_1$ coordinate system, when the axis of division is perpendicular to the $I_1$ axis, the two halves obtained are identified by the direction $+I_1$ (or north or N) and $-I_1$ (or south or S). When the axis of division is perpendicular to $I_0$ axis, the two halves obtained are referred with the direction $+I_0$ (or east or E) and $-I_0$ (or west or W), as shown in Fig. 3.2.

Each node in a binary tree is stored as a record consisting of four fields. The first three fields are pointers to the node's father and to its two sons, i.e., quadrants labeled N(north) and S(south) or E(east) and W(west) depending on the level of the tree. (The root node

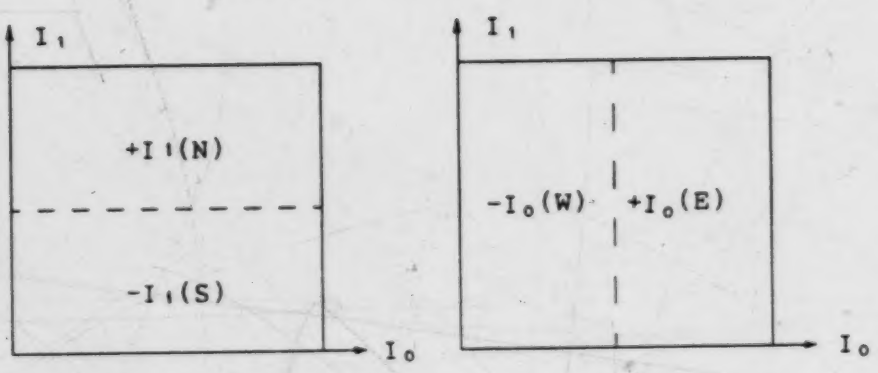



Fig. 3.2. Quadrants labelling in 2-D binary image.

has no father and leaf nodes have no sons). The fourth field represents the node itself. It has the value BLACK if the region represented by the node is all black, similarly it is WHITE when the region is all white, and it is GRAY otherwise. GRAY nodes represent nonterminal nodes in the binary tree, while BLACK and WHITE nodes are leaf nodes, i.e., terminating nodes. Given a node P (node's label) and a son Q (the direction), these fields are referenced as FATHER(P) and SON(P,Q) respectively. Assume that the father of the root node is NULL. The specific quadrant in which a node, say P lies relative to its father can be determined by use of the function SONTYPE(P), which has a value of Q if SON(FATHER(P),Q) = P. In the case of non-binary images, each non terminal node stores the average gray scale value of its two sons.

The following functions are defined to aid in the operations involving the block's quadrants and its boundaries. OPPOSITE(X,Y) (here, X and Y refers to the directions) is a boolean function which returns T or F, depending on whether X is in the opposite direction of Y or not. For example, OPPOSITE(N,S) = T, OPPOSITE(S,W) = F. ADJACENT(X,Y) (here, X refers to the direction and Y refers to the node's label) is a function which provides the mirror image directional movements which are helpful in finding adjacent blocks. For example, ADJACENT(N,S) = N. Table I provides complete values for these two functions for a 2-D image.

Table I

Tables showing the values for OPPOSITE and ADJACENT functions for a 2-D image.

| X \ Y | N | E | S | W |
|---|---|---|---|---|
| N | F | F | T | F |
| E | F | F | F | T |
| S | T | F | F | F |
| W | F | T | F | F |

OPPOSITE(X,Y)

| X \ Y | N | E | S | W |
|---|---|---|---|---|
| N | S | E | N | W |
| E | N | W | S | E |
| S | S | E | N | W |
| W | N | W | S | E |

ADJACENT(X,Y)

In an m-dimensional space of side $2^n$ and coordinate axis $I_0, I_1, \ldots, I_{m-1}$, each node in the binary tree consists of four fields. The first three fields are pointers to the node's father and two sons, i.e., quadrants labeled $+I_i$ and $-I_i$ when the axis of division is perpendicular to the $I_i$th axis. The fourth field is the node value, which is either black, white or gray. The functions OPPOSITE(X,Y) and ADJACENT(X,Y) are defined as follows:

For $i=0,1,\ldots,m-1$,

OPPOSITE($+I_i$,$-I_i$) = T,

OPPOSITE($-I_i$,$+I_i$) = T,

= F, otherwise.

ADJACENT($\pm I_i$,$+I_i$) = $-I_i$,

ADJACENT($\pm I_i$,$-I_i$) = $+I_i$, and

ADJACENT($\pm I_i, +I_j$) = $+I_j$,

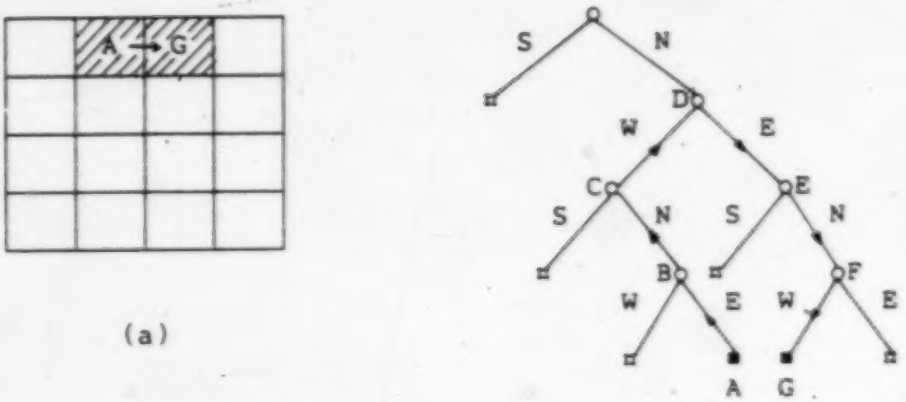
ADJACENT($\pm I_i, -I_j$) = $-I_j$, for $i \neq j$.

With the above concepts, algorithms for some of the mostly used image operations are developed for binary tree data structures.

### 3.2.3 Neighbor Finding Technique

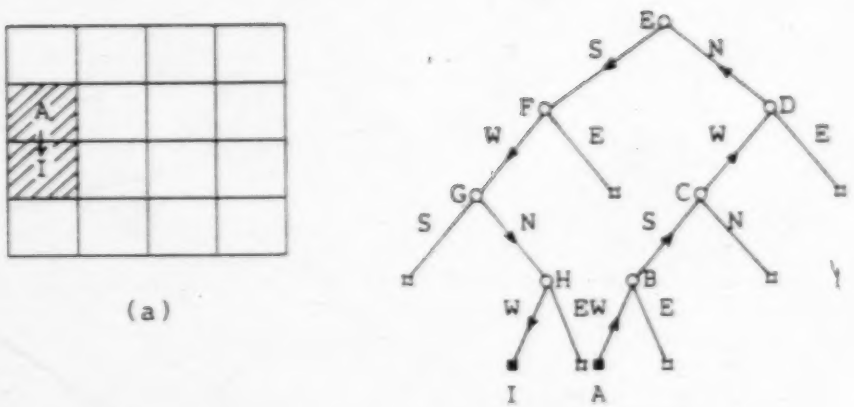
Given any node of a tree corresponding to a specific block of an image, its neighbor of equal size in any given direction is determined by, first locating the common ancestor. For example, to find an eastern neighbor, the common ancestor is the first ancestor which is reached via a west son. Next, retrace the path while making the mirror image moves about an axis formed by the common boundary between the blocks associated with the two nodes. While making the mirror image moves, it should be noted that the direction of the neighbor to be determined is important. If the neighbor required is in the northern or southern direction, the west and east directional moves should be kept the same. Similarly, in finding the western or eastern neighbor, the northern and southern movements should be kept the same.

For example, the eastern neighbor of node A in Fig. 3.3a(a) is G. It is located by ascending the tree until the common ancestor D is found. This requires going through E link to reach B, an N link to reach C and a W link to reach D. Node G is now reached by back tracking along the previous




(a) (b)

Fig. 3.3a. Eastern equal neighbor.




(a) (b)

Fig. 3.3b. Southern equal neighbor.

path with appropriate mirror image moves. This requires descending an E link to reach E, an N link to reach F and W link to reach G.

An algorithm for finding the neighbor of equal size in any given direction D of a given node P is given below. This algorithm is applicable to an image of any dimension.

```
procedure EQUAL_NEI(P,D);
begin
    node : P;
    direction : D;
    return (SON ( if not OPPOSITE(D, SONTYPE(P)) then
        EQUAL_NEI (FATHER(P), D)
        else FATHER(P),
        ADJACENT (D, SONTYPE(P))));
end.
```

It is often the case that neighbors are of different sizes. In such cases, neighboring terminal nodes of equal or greater sizes are of importance. Also the node could be at the border of the image and NULL is returned since there is no neighbor in the specified direction. When the node does not have a neighboring terminal node of equal or greater size, returning a GRAY node of equal size is important. This is because for any given node whose neighbor being sought could have more than one neighboring terminal nodes in the given direction. The algorithm for locating neighbors of

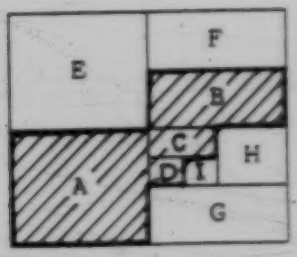
equal or greater size in any direction is given below.

```
procedure EQUAL_GRE_NEI(P,D);
begin
    node : P,Q;
    direction : D;
        if not NULL(FATHER(P))
        and not OPPOSITE(D,SONTYPE(P)) then
        Q:=EQUAL_GRE_NEI(FATHER(P),D)
        else Q:=FATHER(P);
        return(if not NULL(Q) and GRAY(Q) then
        SON (Q,ADJACENT(D,SONTYPE(P)))
        else Q)
end.
```

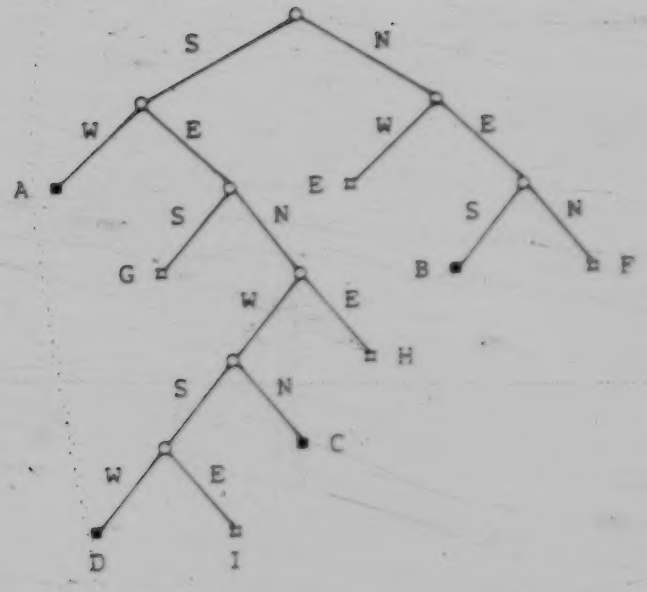
### 3.2.4 Computing Perimeter of Regions

To compute the perimeter of a region represented by a binary tree, traverse the tree in postorder (i.e., the sons of the node are visited first). For each BLACK terminal node, say P, visit all of the nodes whose corresponding blocks have a boundary in common with P's block. These are P's northern, eastern, southern and western adjacencies. For each of the visited nodes that is WHITE, the length of the common boundary is included in the value of the perimeter.

For example, in Fig. 3.4, given a node D, nodes A, G, I and C are visited, of these nodes only I and G are WHITE and




(a)




(b)

Fig. 3.4. An image and its corresponding binary tree.

thus the only contribution to the value of the perimeter is the length of the boundary segments between D and I and D and G.

### 3.2.5 Binary Trees from Binary Arrays

Assume that the given image is a $2^n \times 2^n$ array of unit square pixels, each of which has the value 0 or 1. The binary tree approach to image representation based on successive subdivision of the array into quadrants until blocks which consist entirely of either 1's or 0's are obtained. The method of obtaining a binary tree from a binary array is as follows.

This method of building a binary tree from a binary array examines each pixel in the binary array only once. This is analogous to postorder tree traversal. For example, the pixels in the binary array of Fig. 3.5a are labeled in the order in which they have been examined (e.g., denoting the array by A, we see that A[1,1] is examined first, followed by A[1,2], A[2,1], A[2,2], A[1,3],...). However a node is created only if it is maximal, i.e., if it cannot participate in further merges. For example, Fig. 3.5b shows the nodes in the binary tree resulting from examining pixels of Fig. 3.5a. Note that, since both pixels 1 and 2 are not of the same type, they cannot participate in any further mergers and thus the segment of the binary tree corresponding to their contribution can be constructed. In contrast, pixels 3 and 4 are of the same type (i.e., black)

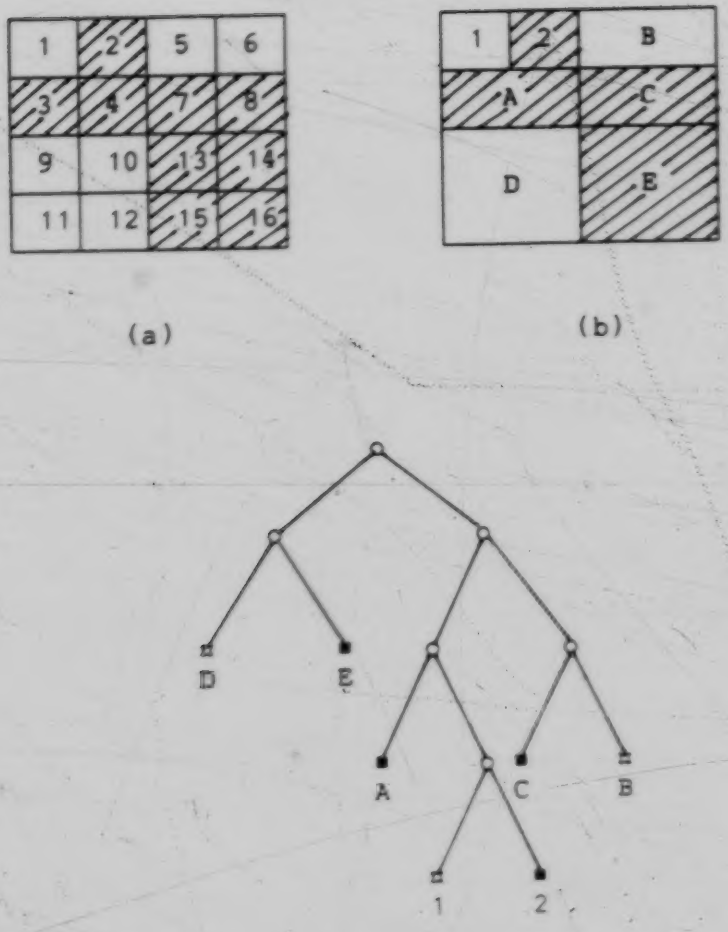



Fig. 3.5. An image, its maximal blocks and its binary tree.

and thus they will be represented by node A in the final binary tree. No nodes are ever constructed corresponding to these pixels. As a final example of the merge, observe that nodes 13, 14, 15, 16 are represented by node E in the binary tree of Fig. 3.5b.

### 3.2.6 Area of the Image Represented by a Binary Tree

This algorithm finds the area of an image represented by a binary tree. This algorithm is based on tree traversal.

```
procedure AREA(P, N, N);
begin
    IMAGE_AREA:integer;
    Q:quadrant;
        IMAGE_AREA:=0;
        if GRAY(P) then
        begin
            if Q in {N, S} then
            IMAGE_AREA:=AREA(SON(P,Q),N,N-1) else
            IMAGE_AREA:=AREA(SON(P,Q),N-1,N-1)
        end; else if BLACK(P) then
        IMAGE_AREA:=IMAGE_AREA+2(2*N);
        return(IMAGE_AREA);
end.
```

### 3.2.7 Set Operations

It is often required to compare two regions in an image and find the union and intersection of these regions. Intersection involves comparing two regions in an image and finding what is common in both of them. An other set operation is complementation. The algorithm for these operations involves tree traversal.

**Complement**

Constructing the complement of an image involves changing black pixels into white and white pixels into black. This is very simple in binary trees and does not change the structure of the tree at all.

Input to the tree is a pointer (P) to the root of the tree.

```
procedure COMPLEMENT(P);
begin
    node : P;
    quadrant : I;
    if GRAY(P) then
        for I in {+I, -I} do
        COMPLEMENT(SON(P, I));
        else if BLACK(P):=WHITE; then
        NODETYPE(P):=WHITE;
        else NODETYPE(P):=BLACK;
end.
```

**Intersection**

This procedure finds the logical AND of the two binary images represented by binary trees. The algorithm works on the principle that, when one tree has a son that is a BLACK leaf, and the other has a corresponding son that is not BLACK, the BLACK leaf is replaced by the corresponding subtree. If one tree has a leaf that is WHITE, the intersection tree will have a corresponding WHITE leaf. Finally, if both trees have GRAY nodes in corresponding positions, the nodes are examined recursively using the same process.

Input to the procedure is a pointer to the root of each of the tree.

```
procedure INTERSECTION(P1, P2);
begin
    node : P1, P2, INTERSECT;
    quadrant : I;
    if BLACK(P1) or WHITE(P2) then
    return(COPY(P2));
    else if BLACK(P2) or WHITE(P1) then
    return(COPY(P1));
    INTERSECT := CREATENODE( );
    for I in {+I, -I} do
    begin
        SON(INTERSECT, I):=INTERSECTION(SON(P1, I)
        SON(P2, I));
```

```
            FATHER(SON(INTERSECT, I)) := INTERSECT;
            end;
        return(INTERSECT);
    end.


    procedure COPY(P);
    begin
        binary tree : P, NEWTREE;
        quadrant : I;
        NEWTREE := CREATENODE( );
        NODETYPE(NEWTREE):=NODETYPE(P);
        for I in {+I, -I} do
        if SON(P, I) := NULL then
        SON(NEWTREE, I) := NULL; else
        begin
            SON(NEWTREE, I) := COPY(SON(P, I));
            FATHER(SON(NEWTREE, I) := NEWTREE;
        end;
        return(NEWTREE);
    end.
```

**Union**

The union algorithm is similar to the intersection algorithm. The trees are traversed in parallel and the decisions are made whenever either of the traversals reaches a leaf. The decisions are the complement of the intersection

algorithm. When a BLACK leaf is encountered, this becomes the subtree at the current position in the union tree. A WHITE leaf in one tree results in the corresponding subtree of the other tree becoming the subtree at the current position of the union tree. When there are two GRAY nodes, the procedure is called recursively for the subtree rooted at these nodes.

Input to the algorithm is a pointer to the root node of each tree.

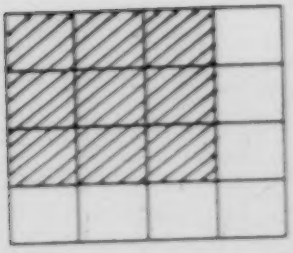
```
procedure UNION(P1, P2);
begin
    node : P1, P2, UNI;
    quadrant : I;
    if BLACK(P2) or WHITE(P1) then
    return (COPY(P2)) else
    if WHITE(P2) or BLACK(P1) then
    return (COPY(P1));
    UNI := CREATENODE( );
    for I in {+I, -I} do
    begin
        SON(UNI, I) := UNION(SON(P1, I), SON(P2, I));
        FATHER(SON(UNI, I)) := UNI;
    end;
    return (UNI);
end.
```

Fig. 3.6 shows the binary trees for the two 2-D binary images, and the binary trees obtained through the intersect and union algorithms and the corresponding images.

### 3.2.8 From Lower to Higher Dimensional Images

Higher dimensional images can be built from lower dimensional images. for example, a line segment can be built from a set of points, a polygon can be built from a set of line segments, etc. An m-dimensional image can be constructed from a set of (m-1)-dimensional images. This is of potential use in the construction of 3-D images from 2-D sectional images, which in turn can be built from the 1-D images or rasters. This is a natural approach for application to *Computed Tomography* (CT) images.

An algorithm to build an m-dimensional tree from a set of (m-1)-dimensional trees uses the divide and conquer technique. Information required to build a tree corresponding to an m-dimensional image is $2^n$ number of trees corresponding to (m-1)-dimensional sectional images, where n is the resolution of digitization. The height of a complete (m-1)-dimensional binary tree is n(m-1)+1. The height of a complete m-dimensional binary tree equals nm+1. The increase in height in going from (m-1)-dimensional trees to an m-dimensional tree is n. Since there are $2^n$ (m-1)-dimensional trees, the binary merging of the trees increases the tree height by $\log 2^n = n$. (merging two trees at a time increases the tree height by 1). Hence it is



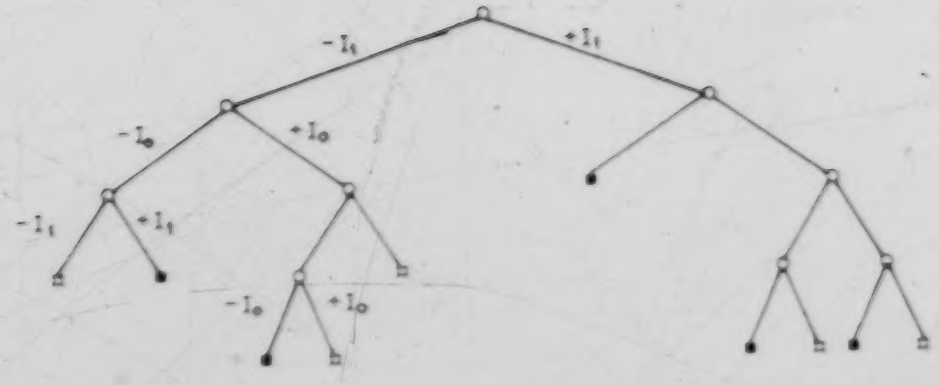
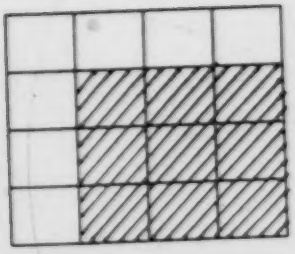



Fig. 3.6a. An image and its binary tree.



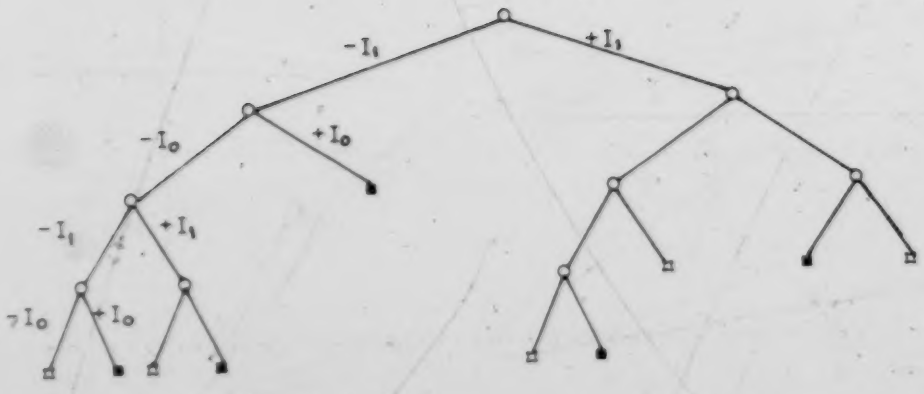
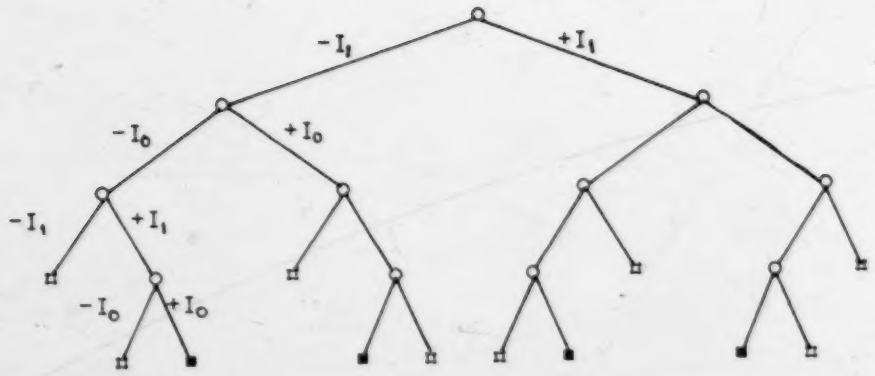



Fig. 3.6b. An image and its binary tree.



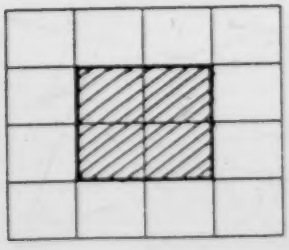
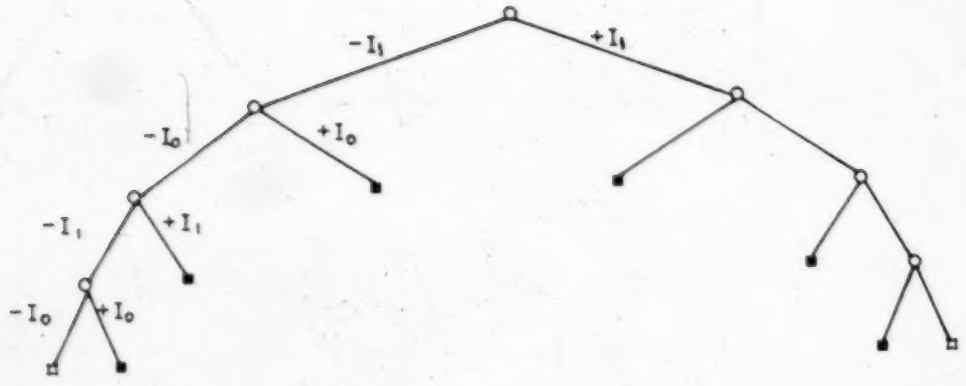



Fig. 3.6c. Binary tree from intersect algorithm and its corresponding binary image.



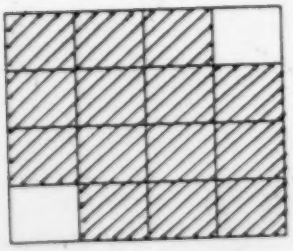



Fig. 3.6d. Binary tree from Union algorithm and its corresponding binary image.

possible to build an m-dimensional tree from a set of (m-1)-dimensional trees.

**Algorithm:**

Let $K=K0,K1,\ldots,K2^{n-1}$, correspond to the $2^n$ number of (m-1)-dimensional trees. The branches at each level of the tree corresponds to $\pm I_i$, where $I_i$ is the axis of division and $i = 1,2,\ldots,m-2$. At each stage of the algorithm, two trees are merged. Merging of two trees consists of the creation of two new nodes and redirection of pointers [see Fig. 3.8]. This increases the tree height by 1. The increased height corresponds to the m-dimensional axis of division, namely $I_{m-1}$. At the end of the first stage of merging, there will be $2^n/2 = 2^{n-1}$ trees corresponding to $K = K01,K23,\ldots,K2^{n-2}2^{n-1}$. In the next stage of merging the trees K01 and K23, K45 and K67, etc, are used. This process is continued until all the trees are merged into one m-dimensional tree.

For example, consider a set of 1-D trees corresponding to n=2. There will be four 1-D trees, say K0, K1, K2 and K3 (see Fig. 3.7a). It is required to build a 2-D tree from this set. Height of the tree for 1-D sections will be 3. Height of the tree corresponding to 2-D image will be 5. At each stage of merging the 1-D trees, the tree height will be increased by 1. Since there are four sections, namely K0, K1, K2 and K3, at first the sections K0 and K1 and then sections K2 and K3 are merged to give two trees of height 4. Let these trees be K01 and K23. In the next stage of merging

K01 and K23 are used and the resultant tree will be the required 2-D tree whose height will be 5. Fig. 3.8 shows the merging and resultant trees at each stage of the algorithm.

When all the trees are merged and formed into one m-dimensional tree, it will be a complete binary tree (Fig. 3.10) having redundant information. To eliminate the redundant information, the tree has to be traversed and whenever a father node has two of its son nodes of same colour, the father node is replaced by a terminal node. The colour of the node will be the same as its son node and the sons are removed.

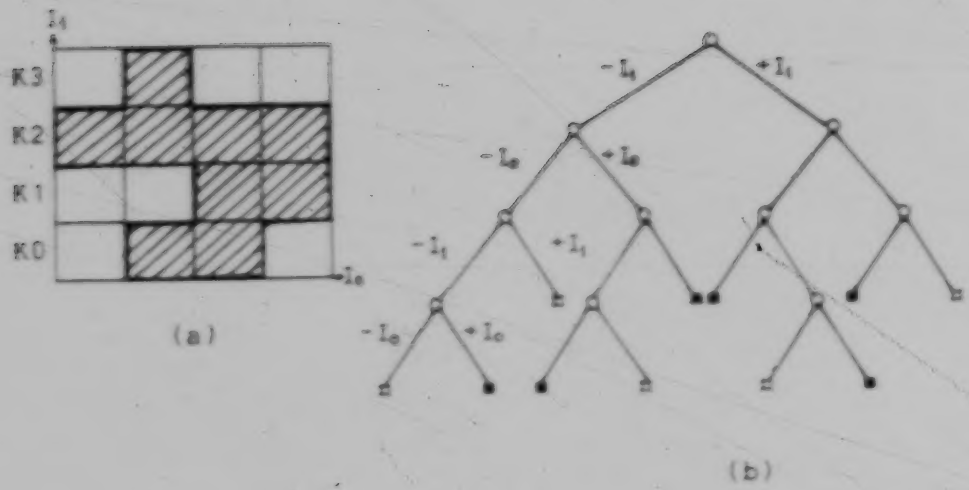



Fig. 3.7. A 2-D image and its binary tree.

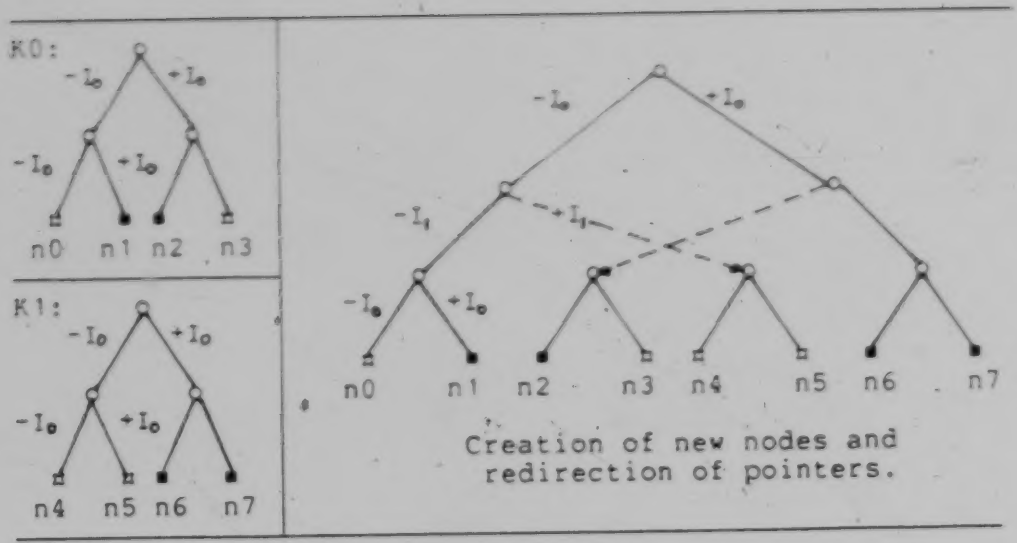



Fig. 3.8. Merging trees K0 and K1.

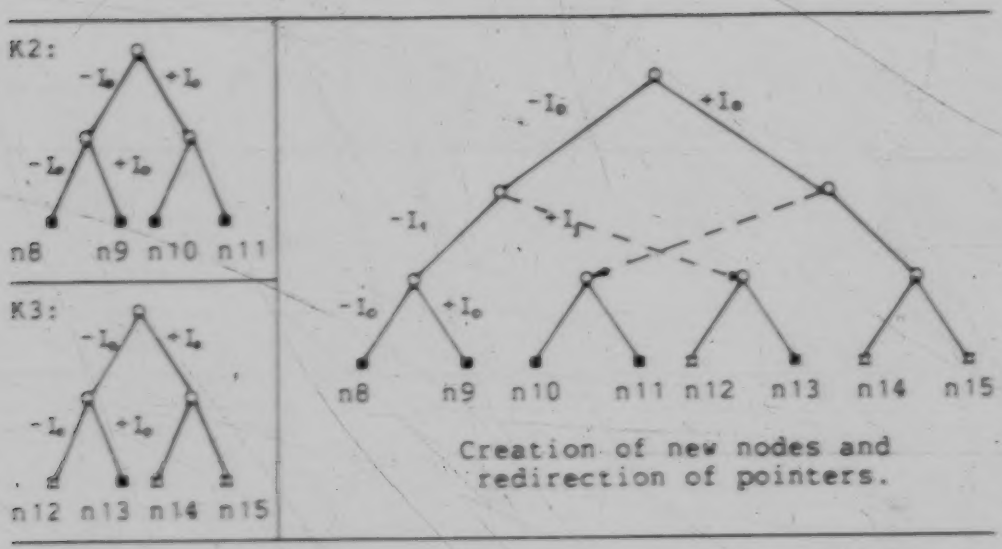



Fig. 3.9. Merging trees K2 and K3.

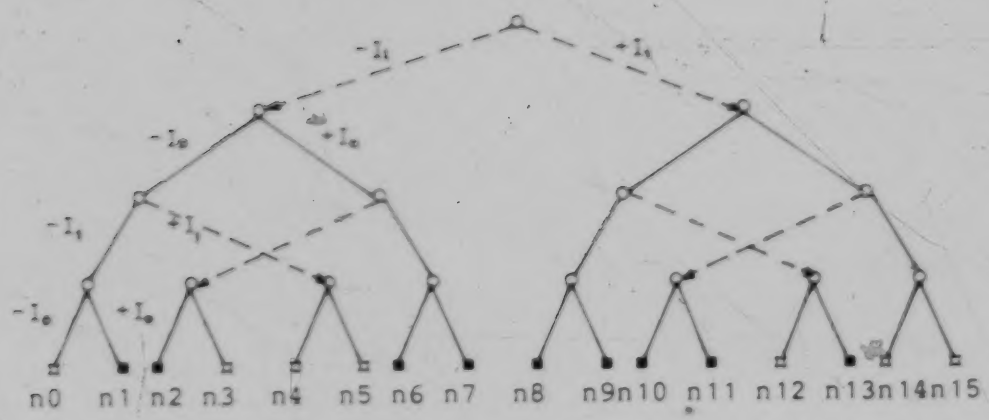



Fig. 3.10. Generation of binary tree of a 2-D image
from the binary trees of its 1-D slices.

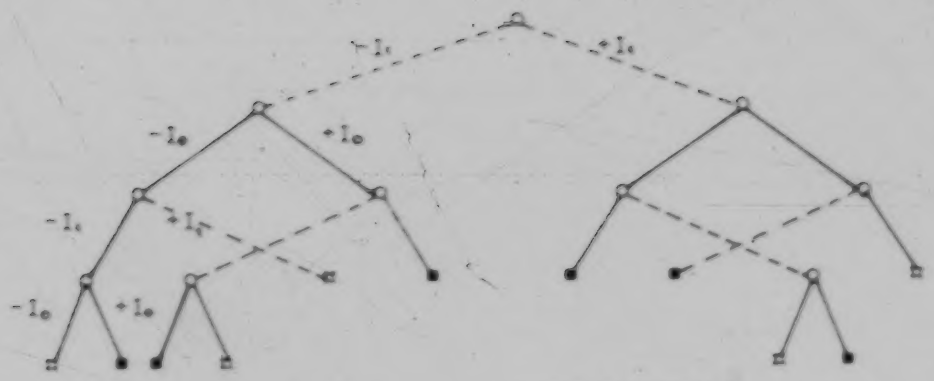



Fig. 3.11. Tree resulting from the traversal and elimination of redundant information in the tree in Fig. 3.10.

## 3.3 ASYMMETRIC BINARY TREES

In the asymmetric binary tree formation, the area is subdivided into rectangular areas, but the subdivision may not necessarily be at the midpoint of the axis. Because the subdivision is variable along the axes, storage can be saved by intelligent selection of the subdivision. For example, if a very small object has to be represented in the middle of a large empty space, the symmetric binary tree subdivision model will have to traverse many levels of the tree before it gets to the object. Using unequal subdivision, the object can be singled out with fewer levels of tree traversal. An asymmetric binary tree formation is shown in Fig. 3.12.

### 3.3.1 Properties of Asymmetric Binary Trees

1. The image can be represented with as few levels of the tree as possible.
2. The asymmetric binary tree nodes have to contain the information regarding the position of the axis of division along with the other regular binary tree information.
3. This type of asymmetric division can be done on 2-D as-well-as 3-D images.
4. Since the image is singled out from the background, the part of the tree representing the image remains the same under linear translation. Hence that part of the tree is shift invariant (see Fig. 3.13).

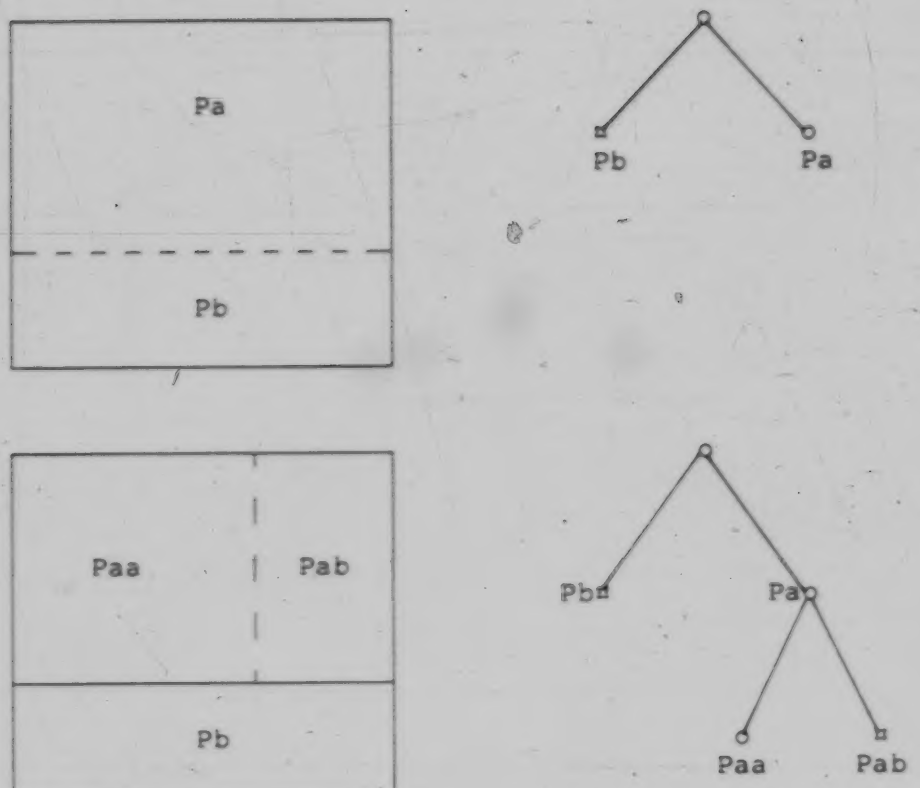



Fig. 3.12. An asymmetric binary tree.

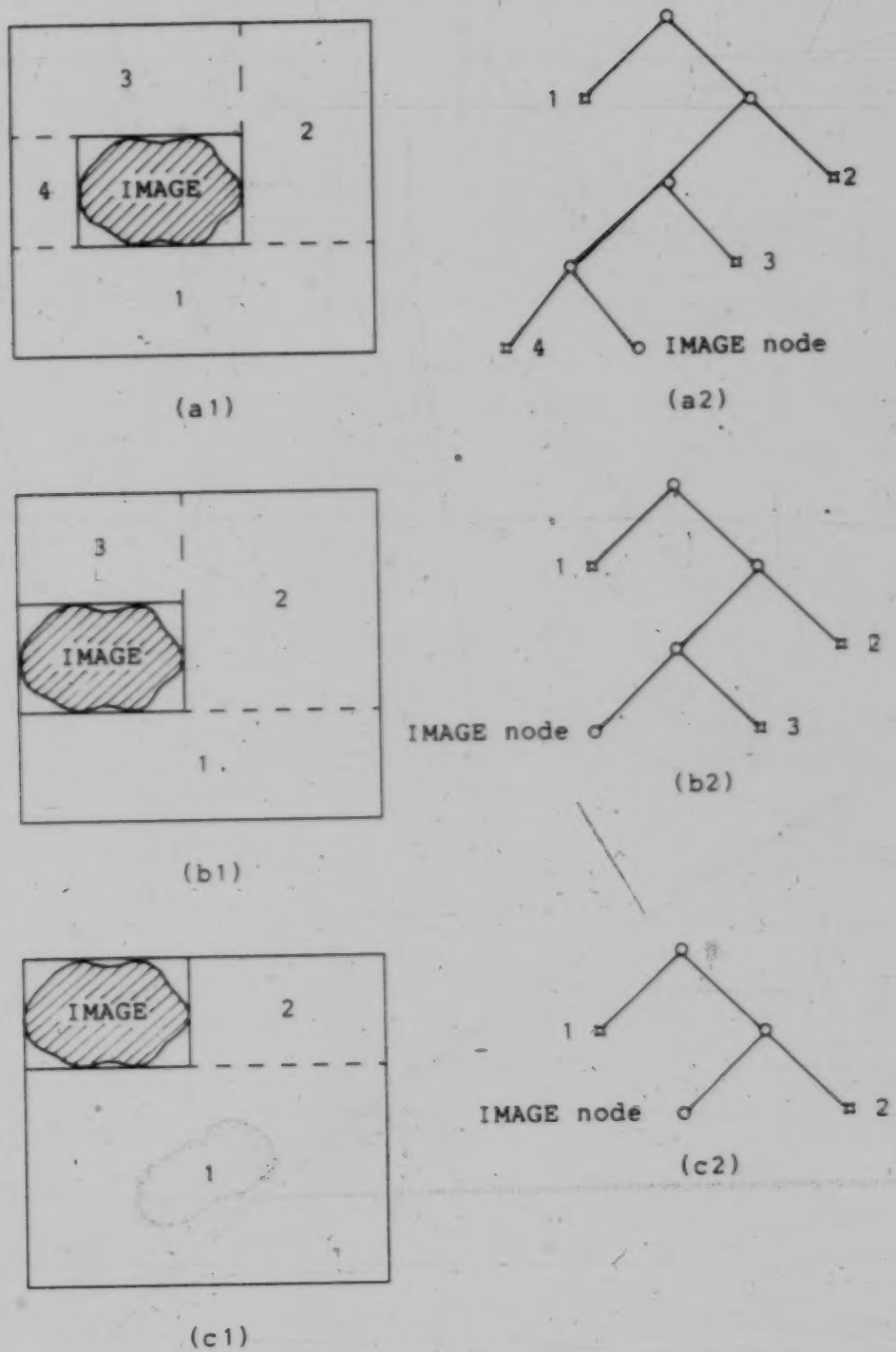



FIG. 3.13. Asymmetric binary tree of an image at different positions in a grid.

### 3.4 LINEAR BINARY TREES

A quadtree may be represented without pointers by encoding each black node with an integer from the set {0,1,2,3}, where the digit reflects successive quadrant subdivision. The sorted array of black nodes is referred to as a *linear quadtree* [16]. It was shown that it introduces a saving of at least 66% of the computer storage required by a regular quadtree. Based on the same principle, linear oct-tree was introduced [18] for 3-D binary image representation. Here, each black node is encoded with a integer from the set {0,1,2,3,4,5,6,7}. Using the same principle, linear binary trees will now be introduced. In this format, each black node is encoded with a binary number from the set {0,1}. This type of encoding is applicable to 2-D images, 3-D images as-well-as higher dimensional images. Futhermore it is better than linear quadtrees or oct-trees in space and time efficiency.

Linear binary trees are basically a result of compaction technique, which can be used for binary valued images. The technique consists of encoding the black pixels depending on their position in the grid, in terms of binary numbers and storing the number in an array. This provides for a substantial savings in the memory requirement (see Chapter 4). The following sections deal with the encoding of black pixels and some of the image operations that can be done using linear binary trees.

All the algorithms presented for linear binary trees are first described for 2-D images and then they are generalized for an m-dimensional space.

### 3.4.1 Encoding of Black Pixels

The first step in linear binary trees is to devise an algorithm to encode black pixels. Consider an array of $2^n \times 2^n$ elements, where black pixels may be randomly selected. If the 2-D image is identified with an x, y coordinate system, each coordinate axis is chosen in order and the area is divided into two halves by an axis perpendicular to the chosen axis. For example, if the y axis is chosen, then an axis is drawn perpendicular to y axis midway along the y axis. The lower part is then encoded with 0 and the upper part is encoded with 1. Next the x axis is chosen and the area is divided into two parts by an axis perpendicular to x axis. Now the left part is encoded with a 0 and the right part is encoded with 1. This procedure is repeated to the pixel level. Then each black pixel is encoded with a weighted binary code, where each successive digit represents the quadrant subdivision from which it originates.

Encoding of a black pixel is basically a mapping of the position of a pixel in $2^n \times 2^n$ array (for a 2-D image) to a binary number K. To find K, let,

$$I_x = I_{x,n-1} I_{x,n-2} \cdots I_{x,0},$$

$$I_y = I_{y,n-1} I_{y,n-2} \cdots I_{y,0},$$

where, $I_{x,i}$, $I_{y,i}$ is either 0 or 1. Thus $I_{x,i}, I_{y,i}$ indicates

to which quadrant the pixel belongs at each level of subdivision. Once the values of $I_x$ and $I_y$ are determined, to get K, interleave the values of $I_x$ and $I_y$. Here, K can be expressed as

$K = K_{n-1}K_{n-2} \ldots K_0$, where $K_i = I_{y_i} I_{x_i}$, hence

$K = I_{y_{n-1}} I_{x_{n-1}} I_{y_{n-2}} I_{x_{n-2}} \ldots I_{y_0} I_{x_0}$.

**Example:** If a pixel in cartesian coordinates is at (6,5) in a $2^3 \times 2^3$ pixel array, then

$I_x = I_{x_2} I_{x_1} I_{x_0} = 110$ and

$I_y = I_{y_2} I_{y_1} I_{y_0} = 101$, hence

$K = K_2 K_1 K_0$,

$= I_{y_2} I_{x_2} I_{y_1} I_{x_1} I_{y_0} I_{x_0}$,

$= 110110$ (see Fig. 3.14).

Hence, the procedure for encoding black pixels:

```
procedure ENCODING (n, Ix, Iy, K)
    n,l:integer; Ix,Iy,K:binary array;
    begin
    for l=n-1 to 0 instep -1 do
        begin
        K(l) = Iy(l)Ix(l)
        end;
    end.
```

In general, for an m-dimensional space of 2"x2"x ... x2" (m elements),

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 101 010 | 101 011 | 101 110 | 101 111 | 111 010 | 111 011 | 111 110 | 111 111 |
| 101 000 | 101 001 | 101 100 | 101 101 | 111 000 | 111 001 | 111 100 | 111 101 |
| 100 010 | 100 011 | 100 110 | 100 111 | 110 010 | 110 011 | 110 110 | 110 111 |
| 100 000 | 100 001 | 100 100 | 100 101 | 110 000 | 110 001 | 110 100 | 110 101 |
| 001 010 | 001 011 | 001 110 | 001 111 | 011 010 | 011 011 | 011 110 | 011 111 |
| 001 000 | 001 001 | 001 100 | 001 101 | 011 000 | 011 001 | 011 100 | 011 101 |
| 000 010 | 000 011 | 000 110 | 000 111 | 010 010 | 010 011 | 010 110 | 010 111 |
| 000 000 | 000 001 | 000 100 | 000 101 | 010 000 | 010 001 | 010 100 | 010 101 |

Fig. 3.14. Encoding of black pixels in a linear binary tree.

$$I_1 = I_{0,n-1}I_{0,n-2} \ldots I_{0,0},$$

$$I_2 = I_{1,n-1}I_{1,n-2} \ldots I_{1,0},$$

$$\vdots$$

$$I_m = I_{m-1,n-1}I_{m-1,n-2} \ldots I_{m-1,0}.$$

and K can be expressed as $K = K_{n-1}K_{n-2} \ldots K_0$, where

$$K_i = I_{0,i}I_{1,i} \ldots I_{m-1,i}.$$

```
procedure ENCODING (n, I0,I1,...,Im-1, K)
    n:integer; I0,I1,...,Im-1,K:binary array;
    begin integer:l;
    for l=n-1 to 0 instep -1 do
        begin
        K(l) = I0(l)I1(l)...Im-1(l)
        end;
    end.
```

Once all the black pixels are encoded, sort them in an array.

### 3.4.2 Adjacency

With this structure, an algorithm for finding the adjacent pixel in any given direction is given below. Let $K = K_{n-1}K_{n-2}\ldots K_0$ be a binary number representing a pixel, and let $S = S_{n-1}S_{n-2}\ldots S_0$ be the adjacent pixel in the required direction.

Here, $K_l = I_{y,l}, I_{x,l}$ and

$$I_x = I_{x,n-1}I_{x,n-2}\ldots I_{x,0},$$

$$I_y = I_{y,n-1}I_{y,n-2}\ldots I_{y,0}.$$

To find an adjacent pixel in the northern direction (i.e., increasing Y-axis), add 1 to $I_y$

$I'_y = I_y+1$ (binary addition) and let $I'_y$ be

$$I'_y = I'_{y,n-1}I'_{y,n-2}\ldots I'_{y,0}.$$

The adjacent pixel in the northern direction is given by S, where,

$S = S_{n-1}S_{n-2}\ldots S_0$ and

$S_i = I'_{y,i}I_{x,i}$.

```
procedure N-ADJACENT (n, K, S)
    n:integer; K,S:binary array;
    begin
        get I_x,I_y;
        compute I'_y = I_y+1; (binary addition)
        get S, where S_i =I'_{y,i}I_{x,i}
    end.
```

The processing required to find the adjacent pixels in the other directions is given below,

Adjacent pixel in decreasing Y-direction:

$I'_y = I_y-1$; and $S_i = I_{x,i}I'_{y,i}$.

Adjacent pixel in increasing X-direction:

$I'_x = I_x+1$; and $S_i = I'_{x,i}I_{y,i}$.

Adjacent pixel in decreasing X-direction:

$I'_x = I_x-1$; and $S_i = I'_{x,i}I_{y,i}$.

In general, consider a m-dimensional space, to find the adjacent hypervoxel S of a given hypervoxel K, where

$K = K_{n-1}K_{n-2}\ldots K_0$,

$K_i = I_{0,i}I_{1,i}\ \ldots\ I_{m-1,i}$,

$$I_1 = I_{0,n-1}I_{0,n-2} \ldots I_{0,0},$$

$$I_2 = I_{1,n-1}I_{1,n-2} \ldots I_{1,0},$$

$$\vdots$$

$$I_m = I_{m-1,n-1}I_{m-1,n-2}\ldots I_{m-1,0}.$$

To find adjacent hypervoxel in jth direction, get $I_j$ where,

$I_j = I_{j,n-1}I_{j,n-2}\ldots I_{j,0}$.

compute $I'_j = I_j + 1$;

then adjacent hypervoxel is given by S, where

$S = S_{n-1}S_{n-2}\ldots S_0$, and

$S_i = I_{0,i}I_{1,i} \ldots I'_{j,i} \ldots I_{m-1,i}$

### 3.4.3 Linear Translation

An algorithm to translate the given image in any given direction by d units is given below.

It is clear that the only code that would change would correspond to the direction of translation. So, given a binary number K representing a pixel, it would change to T due to linear translation by d units. Where,

$$K = K_{n-1}, K_{n-2}, \ldots, K_0,$$

$$K_i = I_{y,i}, I_{x,i},$$

$$I_x = I_{x,n-1}I_{x,n-2},\ldots,I_{x,0},$$

$$I_y = I_{y,n-1}I_{y,n-2},\ldots,I_{y,0}.$$

To translate the image in the northern direction (increasing y-axis) by d units, when compute for all pixels $I'_y$ where, $I'_y = I_y + d$ (binary addition).

Let $I'_y = I'_{y,n-1}I'_{y,n-2}, \ldots, I'_{y,0}$.

So, each pixel has to be translated to T where,

$$T = T_{n-1}T_{n-2}, \ldots, T_0,$$

and $T_i = I'_{y,i}I_{x,i}$.

```
procedure N-TRANSLATION (n, K, T, d)
    n:integer; K,T:binary array; d:binary number;
    begin
        get I_x, I_y.
        compute I'_y = I_y+d; (binary addition)
        get T, where T_i = I'_{y,i} I_{x,i}
    end.
```

Processing required to translate the image in the other directions is given below.

To translate in decreasing Y-direction by d units,

$$I'_y = I_y - d; \text{ and } T_i = I'_{y,i}I_{x,i}.$$

To translate in increasing X-direction by d units,

$$I'_x = I_x + d; \text{ and } T_i = I_{y,i}I'_{x,i}.$$

To translate in decreasing X-direction by d units,

$$I'_x = I_x - d; \text{ and } T_i = I_{y,i}I'_{y,i}.$$

In general, consider an m-dimensional space, with each hypervoxel K, where,

$$K = K_{n-1}K_{n-2} \ldots K_0,$$

$$K_i = I_{0,i}I_{1,i} \ldots I_{m-1,i},$$

$$I_1 = I_{0,n-1}I_{0,n-2} \ldots I_{0,0},$$

$$I_2 = I_{1,n-1}I_{1,n-2} \ldots I_{1,0},$$

$$\vdots$$

$$I_m = I_{m-1,n-1}I_{m-1,n-2} \ldots I_{m-1,0}.$$

To determine the position of the hypervoxel K when it is translated by d units in the jth direction, get $I_j$ where,

$$I_j = I_{j,n-1}I_{j,n-2} \ldots I_{j,0},$$

compute $I'_j = I_j + 1$ (binary addition), then the new hypervoxel position is given by:

$T = T_{n-1}T_{n-2} \ldots T_0$, where

$T_i = I_{0,i}I_{1,i} \ldots I'_{j,i} \ldots I_{m-1,i}$.

### 3.4.4 Rotation

An algorithm to rotate the given image in any direction by a multiple of ±90° is given below.

Suppose (x,y) is a point in cartesian coordinate system, the new point after rotation by angle $\alpha$ is given by $(x',y')$ where,

$$[x' \; y'] = [x \; y] \begin{bmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{bmatrix} ; \text{ where } \alpha = 90°$$

Hence, $[x' \; y'] = [-y \; x]$.

When $\alpha = 180$, then $[x' \; y'] = [-x \; -y]$.

Let K be the initial position of a pixel where $K = K_{n-1}K_{n-2}\ldots K_0$. and it would change to R in the new

position after rotation, where

$R = R_{n-1}R_{n-2} \ldots R_0$. Here,

$K_i = I_{x,i}, I_{y,i}$,

$I_x = I_{x,n-1}I_{x,n-2}, \ldots, I_{x,0}$,

$I_y = I_{y,n-1}I_{y,n-2}, \ldots, I_{y,0}$.

To rotate the image in the counter-clockwise direction (from X towards Y) about the point at the center of the grid at $2^{n-1}x2^{n-1}$.

Take $I'_y$ as $I_x$ i.e., $I'_y = I_{x,n-1}I_{x,n-2}\ldots I_{x,0}$.

and $I'_x$ is computed from $I_y$

$I'_x = I'_{y,n-1}I'_{y,n-2} \ldots I'_{y,0}$. where $I'_{y,i}$ is the inverse of $I_{y,i}$.

Now the new position of the pixel in the rotated position is given by R where,

$R = R_{n-1}R_{n-2} \ldots R_0$, $R_i = I'_{y,i}I'_{x,i}$.

**Example:** Consider a pixel in $2^3x2^3$ grid, in which $I_x = 101$ and $I_y = 110$. The pixel in the rotated position would be

$I'_y = I_x = 101$,

$I'_x = I'_y = 001$, and

$S_i = 100011$ (see Fig. 3.14)

Now consider an m-dimensional space, when the image is rotated about a particular axis, only the coordinate values corresponding to the plane orthogonal to the axis of rotation would change as described in the 2-D case. Suppose the image is rotated about the jth axis and if $I_k$ and $I_i$ are the axes forming the plane orthogonal to the axis of rotation, then depending on the axis of rotation only, these

two coordinate values would change as described in 2-D cases.

To rotate the image about 180° is very simple. If K is the binary number representing a pixel. The binary number R in the rotated position is given by inverting all the digits in K, i.e., $R_i = K'_i$.

**Example:** consider a pixel 110110 in a $2^3 \times 2^3$ grid, in which $I_x = 101$ and $I_y = 110$. The pixel in the rotated position would be

$I'_x = 010$,

$I'_y = 001$, and

$S_i = 001001$ (inversion of 110110).

### 3.4.5 Set-theoretic Operations

The most extensively used set-theoretic operations, namely the union and intersection are easy to do with this structure. For finding the union of two linear binary trees, merge both the lists, after merging if there are any duplication of nodes, only one is stored. To get the intersection of two linear binary trees, find the common nodes of two trees, and store those values.

# CHAPTER 4

## EVALUATION

In this chapter, data formats of *binary trees* and *linear binary trees* are evaluated and compared with other data formats for their space and time efficiencies.

The most important feature of spatial data structures is storage efficiency. Since the data base could be in the order of gigabytes, the prime consideration has to be the actual number of bytes required by the file. It is difficult to analytically establish the space efficiency of different data structures, because each one is efficient in storing certain kinds of images. Nevertheless, choosing a representative image under a given resolution will give some idea about the space requirements of different data structures. In this section, a region of size $2^k \times 2^k$ appearing anywhere in a $2^n \times 2^n$ image is considered. Both worst and best positions are identified and the space efficiency is determined for each one.

Other important considerations in efficiency are the general procedures to compute spatial properties such as area and perimeter calculations, to perform spatial manipulations such as set theoretic operations, overlays, windowing and browsing. In this thesis, set theoretic operations are discussed in detail.

## 4.1 SPACE EFFICIENCY

Consider a binary tree data format as defined in Chapter 3. Let N be the total number of nodes, B the number of BLACK nodes, W the number of WHITE nodes and G the number of GRAY nodes. The following relations are valid for binary trees, (see 24):

$$N = 2G + 1 = 2(B+W) - 1,$$

$$B = G - W + 1,$$

$$W = G - B + 1,$$

$$G = B + W - 1.$$

In the following sections, a binary tree of a $2^n x 2^n$ image in which a $2^k x 2^k$ region occurs, where $k < n$, is considered. For this image, the best and worst case values of N, B, W and G are computed as a function of n and k.

### 4.1.1 Best Case

The best case occurs when the region can be represented by a single node at level 2k. This means that there are no nodes at levels 0 through 2k-1, two nodes at each of levels 2k through 2n-1 and one node at level 2n. Thus,

$$N = 2(2n - 2k) + 1 = 4(n - k) + 1.$$

Of these one gray node occurs at each level from 2k+1 through level 2n and a white node at each level from 2k through 2n-1. Thus,

$$B = 1,$$

$$W = (2n - 2k) = 2(n - k),$$

$$G = (2n - 2k) = 2(n - k).$$

This occurs whenever the position (i, j) of the region (see Fig. 4.1) is such that i mod $2^k$ = 0 and j mod $2^k$ = 0, thus only $O(n-k)$ nodes are required when the region is in any of $2^{2(n-k)}$ positions in the image. In any other position G would increase since there would now exist black nodes at levels less than 2k and there must be one GRAY node at each level except the lowest level in any binary tree. Hence $G \geq 2(k-n)$. But $N = 2G+1$, thus there can be no position of the region which results in a binary tree containing fewer than $4(n-k)+1$ nodes.

### 4.1.2 Worst Case

Consider the case where the region is at position (i, j) such that i mod $2^k$ = 1 and j mod $2^k$ = 1, i.e., the region is shifted to the right and down one pixel from the best case position (Fig. 4.2). In this case there is a border of level 0 black blocks along the left, level 1 black blocks along at the top side of the region. Inside these two borders there is a column and rows of level 2 and 3 blocks, and so on. This continues until a single black block occurs. Finally a band of black blocks at level 0 and 1 fills the right and bottom border of the region. Thus the number of black blocks is given by:

$$B = \sum_{i=1}^{k-1} (2^{k-i}-1) + 2^k + 2^k + 2^{k-1}-1,$$

$$= 9 \times 2^{k-1} - 2k - 3.$$

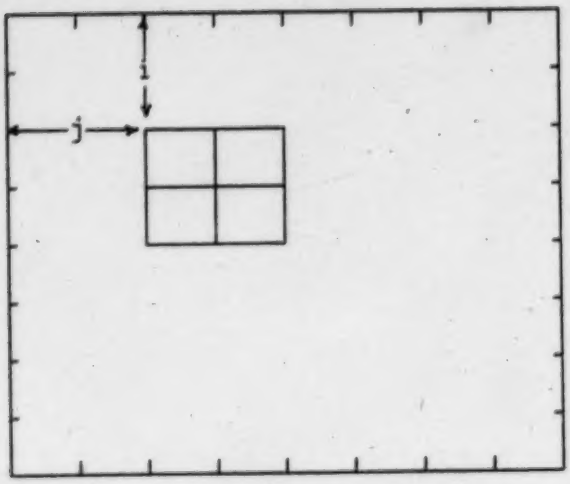



Fig. 4.1. Best case position of the region.

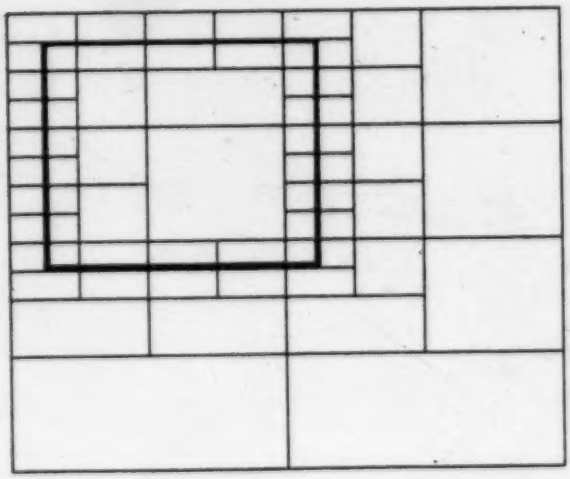


Fig. 4.2. Worst case position of the region.

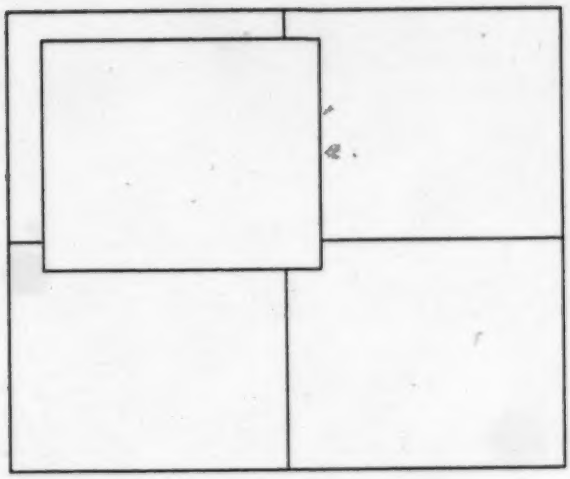


Fig. 4.3. A $2^k \times 2^k$ region completely covered by $2^{k+1} \times 2^{k+1}$ region.

To compute the number of white nodes W, consider a $2^{k-1}x2^{k-1}$ segment which can completely cover the region (Fig. 4.3). The number of white nodes W' of such a region represented by binary a tree can be readily computed:

$$W' = 1 + \sum_{i=0}^{k-1} 2^i + 2\sum_{i=0}^{k-1} 2^i + 2k + 2^{k-1} + 2^k - 1,$$

$$= 9x2^{k-1} + 2k - 3.$$

Since this $2^{k-1}x2^{k-1}$ block completely includes the region, all other WHITE nodes must be at levels no less than 2k+2. Therefore W=W'+W" where W" is the number of nodes at level 2k+2 through 2n. At best the $2^{k-1}x2^{k-1}$ block can be represented as a single gray node at level 2k+2. In this case there will be exactly one gray node at level 2n, one gray node and one white node at each of the levels 2n-1 through 2k+2, and four gray nodes corresponding to the $2^kx2^k$ block at level 2k. Thus at best:

$$W = W' + \sum_{i=2k+2}^{2n-1} 1,$$

$$= 9x2^{k-1} + 2k - 3 + (2n - 1 - 2k - 2 + 1),$$

$$= 9x2^{k-1} + 2n - 5.$$

At worst, the region is so placed that the lower right corner of the region just crosses over the center of the image, i.e., the region is at a position $(i, j) = (2^{n-1}-2^k+1, 2^{n-1}-2^k+1)$. Here there will be one gray node at level 2n, 2 gray nodes at level 2n-1, 4 gray nodes at level 2n-2, and 4 gray nodes and 4 white nodes at each of the levels 2n-3 through 2k. Thus, at worst:

$$W = W' + \sum_{i=2k}^{2n-3} 4,$$

$$= 9x2^{k-1} + 2k - 3 + 4(2n - 3 - 2k + 1),$$

$$= 9x2^{k-1} + 8n - 6k - 11.$$

From the relationship between N, G, B and W:

$$G = B + W + 1,$$

$$= 9x2^{k-1} - 2k - 3 + 9x2^{k-1} + 8n - 6k - 11,$$

$$= 9x2^{k} + 8(n - k) - 14.$$

$$N = 2G + 1,$$

$$= 9x2^{k-1} + 16(n - k) - 29.$$

Table II shows the complete values for B, W, G and N for the best and the worst case.

Table II

The best and worst case values for B, W, G and N nodes for a binary tree.

| Node Type | Best case | Worst case |
|---|---|---|
| Black (B) | 1 | $9x2^{k-1}-2k-3$ |
| White (W) | 2(n-k) | $9x2^{k-1}+8n-6k-11$ |
| Gray (G) | 2(n-k) | $9x2^{k}+8(n-k)-14$ |
| Total (N) | 4(n-k)+1 | $9x2^{k-1}+16(n-k)-29$ |

### 4.1.3 Comparison

In this section the regular binary tree and the linear binary tree formats are compared with other data formats like arrays, chain codes, quadtrees, linear quadtrees, run-length encoding, and strip trees. In the binary tree representation of a $2^n x 2^n$ image containing a single $2^k x 2^k$ region, in the best case the total number of nodes is $O(n-k)$, and in the worst case the total number of nodes is $O(2^{k+2}+n-k)$.

Each field in a binary tree has three pointer fields containing the addresses of its father and two sons. Since a binary tree may be complete, $2n+1$ bits are required for each of these fields. Two bits are sufficient to represent each node's type. Hence each node uses $3(2n+1)+2 = 6n+5$ bits to store its information. This means that in the worst case $P(n,k)=(6n+5)(9x2^{k+1}+16(n-k)-29)$ bits are required to represent a binary tree. Since P is a monotonically increasing function of both n and k, where $0 \le k \le n$, an upper bound on the size of the binary tree for a given n is $P(n,n-1)=(6n+5)(9x2^{n+1}-13) < nx2^{n+7}$.

The array representation of a $2^k x 2^k$ region in a $2^n x 2^n$ image always requires exactly $2^{2n}$ bits. Thus for large n, a binary tree representation is more space efficient than the array representation, since $nx2^{n+7} < 2^{2n}$.

For example, if n=10, 1048.5K bits are required in the array representation. If k=8, at most P(10,8)=300K bits are required to store the binary tree.

Consider a run length representation of a region. If the region is at a position (i,j) then first i and last $2^n-2^k-i$ rows will have $2^n$ of 0's and the other $2^k$ rows will have j of 0's, $2^k$ of 1's, $2^n-2^k-j$ of 1's. So 3n bits are sufficient to represent each row. Hence $3nx2^n$ bits are required to store the entire image. When $n >> k$, the binary tree representation is more efficient than run length encoding, otherwise both are comparable.

Consider a chain coding representation of a region with a 4 direction chain coding scheme. To store the $2^kx2^k$ image, it is required to specify the starting position and the succession of links which describes the region boundary. 2n bits are required to specify the starting position and 2 bits per link, hence $2^{k+3}$ bits are required to encode the boundary. Therefore $2^{k+3}+2n$ bits are used to encode a region boundary. Thus chain coding is superior to binary tree format by a factor of n.

The primary advantage of chain coding is its compactness and analytical capabilities. The primary disadvantage of chain coding is that no spatial relationships are retained. Also, coordinate transformations, in particular rotation is more difficult with chain codes.

Skeletons are shown to be comparable to chain coding in storage requirements [33]. At the same time, it can have a significant advantage for certain type of region processing problems, such as determining whether a given point is

inside a given region or not, finding the intersection of two or more regions.

Consider a quadtree representation of a region. The best and worst case values of B, W, G and N for a quadtree is given in Table III [from ref 9].

Table III

The best and worst case values for B, W, G and N nodes for a quadtree.

| Node Type | Best case | Worst case |
|---|---|---|
| Black (B) | 1 | $3x2^{k+1}-3k-5$ |
| White (W) | 3(n-k) | $3x2^{k+1}+12n-9k-15$ |
| Gray (G) | (n-k) | $4x2^{k}+4(n-k)-7$ |
| Total (N) | 4(n-k)+1 | $8x2^{k+1}+16(n-k)-27$ |

Both quadtrees and binary trees are comparable in space requirements for a given n and k. In the worst case a binary tree representation requires 12.5% more space than a quadtree representation.

Linear binary tree representations of regions are more efficient compared to linear quadtrees. Since, only the

black nodes are stored in either linear binary tree or linear quadtee formats. In the worst case, the number of black nodes in a binary tree format is $9x2^{k-1}-2k-3$. The number of black nodes in a quadtree format is $12x2^{k-1}-3k-5$. Hence there will be a 25% memory saving by using a linear binary tree format. Also it is clear that a linear binary tree format is superior to a linear oct-tree format in storage efficiency.

Consider a strip tree representation of a region. Strip trees are most efficient in representing straight lines. Assuming that the boundary of the region is curved, then each node in the strip tree requires 8n bits. In the worst case there could be $2^{k-2}$ terminal nodes. Hence there will be a total of $2^{k-3}-1$ nodes. Therefore $8nx2^{k-3}-8n$ bits are required to represent a strip tree. Hence a binary tree format is superior to a strip tree.

In the case of strip trees, intersections and point inside a closed curve calculations can be efficiently done. Curves can be efficiently encoded and displayed at various resolutions. Intersections and union operations can be done under different resolutions. Strip trees require large overhead in terms of space.

### 4.1.4 Time Efficiency

Considering time efficiency of the different algorithms presented in Chapter 3, the intersection algorithm takes time proportional to the traversal of the smallest tree. The

union algorithm takes time proportional to that of the second largest tree. The complement algorithm has time linear in the number of nodes of the tree on which it operates. In the case of neighbor finding algorithms, in the worst case the number of nodes visited is twice the height of the tree.

In the case of a linear binary tree, encoding a black pixel requires $O(n)$ time. This is the same order as in the case of linear quadtrees, but the actual time required is 50% of what is required in linear quadtrees, because the region is encoded with only 0's and 1's instead of 0's, 1's, 2's and 3's. Finding the adjacent pixel in any given direction, linear translation in any given direction and rotation by ±90° requires $O(n)$ time.

Looking into other considerations: windowing and browsing are simple with binary trees. If the intermediate nodes have an average gray scale value of two of their sons, then browsing can be done at different resolutions just by traversing the tree. A mechanism can be built to handle the required part of the image to be displayed at the required resolution.

Since each intermediate node represents a different portion of the image, windowing can be easily done by chosing proper nodes in the tree.

With the above characteristics, the binary tree format can be a good alternative to quadtree and oct-tree formats.

# CHAPTER 5

## CONCLUSION AND FUTURE WORK

In this thesis, the data formats of binary trees and linear binary trees have been introduced as generalized data formats for spatial data handling. This is a hierarchical data format having all the advantages of a hierarchical data format. Its use in multidimensional data processing is established.

For a given resolution of digitization, say n, binary tree data formats are comparable to that of quadtrees in space and time efficiency. In the best case a binary tree format is as efficient as a quadtree and in the worst case, binary trees take only 12.5% more space than quadtrees. It is shown that a neighbor finding technique is universal and hence could be used for images of any dimension. In the worst case, the number of nodes visited in finding a neighbor is twice the height of the tree, i.e., $O(\log n)$. Set theoretic operations are shown to be efficient. The intersection algorithm takes time proportional to the traversal of the smallest tree. The union algorithm takes time proportional to that of the largest tree. The complement algorithm has time proportional to the number of nodes of the tree on which it operates.

It is shown that linear binary trees are very efficient compared to that of linear quadtrees and linear oct-trees. Linear binary trees give 25% memory saving compared to linear quadtrees. It is intuitively clear that linear binary trees are even more efficient when compared to that of linear oct-trees. The encoding of black pixels, linear translation in any given direction, finding an adjacent pixel in any given direction, rotation by ±90° are simple compared to that of either linear quadtrees or linear oct-trees.

As an application of binary tree format, a method of building an n-dimensional image from a set of (n-1)-dimensional images is given. This has potential use in medical image processing.

To judge the effectiveness of this data format over the other ones, it is required to implement each algorithm and to run them on a representative set of test data. From this, a practical evaluation of the binary tree data format can be done and its effectiveness over the other data formats can be established.

## REFERENCES


1. N. Ahuja, "On approach to polygonal decomposition for hierarchical image representation", *Computer Vision, Graphics and Image Processing*, Vol. 24, 1983, pp. 214-220.
2. A.P. Ambler, H.G. Barrow, C.M. Brown, R.M. Burstall and R.J. Popplestone, "A versatile system for computer controlled assembly", *Artificial Intelligence*, Vol. 6(2), 1975, pp. 129-156.
3. D.H. Ballard, "Strip trees: A hierarchical representation for curves", *Communications of the ACM*, Vol. 24(5), 1981, pp. 310-321.
4. D.H. Ballard and C.M. Brown, "Computer Vision", Prentice-Hal Inc. Publication, 1982.
5. J.L. Bentley, "Multidimensional binary search trees used for associative searching", *Communications of the ACM*, Vol. 18, Sept 1975, pp. 509-517.
6. W. Burton, "Representation of many sided polygons and polygonal lines for rapid searching", *Communications of the ACM*, Vol. 29, March 1977, pp. 166-171.
7. W.A. Davis and X. Wang, "A new approach to linear quadtrees", *Technical Report TR 84-9, Department of Computing Science, University of Alberta*, Nov 1984.
8. D.H. Douglass and T.K. Peuker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature", *Canadian Cartographer*, Vol. 10, No. 2, Dec. 1973, pp. 112-122.

9. C.R. Dyer, "Space efficiency of quadtrees", *Computer Graphics and Image Processing*, Vol. 19, 1982, pp. 335-348.

10. C.R. Dyer, A. Rosenfeld and H. Samet, "Region representation: boundary codes from quadtrees", *Communications of the ACM*, Vol. 23, 1980, pp. 335-348.

11. C.M. Eastman, "Representation for space planning", *Communications of the ACM*, Vol. 13, 1970, pp. 242-250.

12. R.A. Finkel and J.L. Bentley, "Quadtrees: a data structure for retrieval on composite keys", *ACTA-Informatica*, Vol. 4, 1974, pp. 1-9.

13. H. Freeman, "Computer processing of line-drawing images", *Computing Surveys*, Vol. 6(1), March 1974, pp. 57-97.

14. H. Freeman, "Application of the generalized chain scheme to map data processing", *Proceedings Conference on Pattern Recognition and Image Processing*, IEEE Computer Society Publ. 1978, pp. 220-226.

15. H. Freeman and P.J. Frauenknecht, "An Efficient Computer Representation Scheme For Linear Map Data", *Proceedings Conference on Pattern Recognition and Image Processing*, IEEE Computer Society Publ. 1982. pp. 315-320

16. I. Gargantini, "An efficient way to represent quadtrees", *Communications of the ACM*, Vol. 25, 1982, pp. 905-910.

17. I. Gargantini and Z. Tabakman, "Linear quad and oct-trees: their use in generating simple algorithms for

image processing", *Graphics Interface 1982*, pp. 123-127.

18. I. Gargantini, "Linear oct-trees for fast processing of three-dimentional objects", *Computer Vision, Graphics and Image Processing*, Vol. 20, 1982, pp. 365-374.
19. R. Gillespie and W.A. Davis, "Tree data structures for graphics and image processing", *Proc. Canadian Man Computer Communication Society Conference*, Waterloo, Ontario, 1981, pp. 155-162.
20. G.M. Hunter and K. Steiglitz, "Operations on images using quadtrees", *IEEE Transaction on PAMI*, Vol. 1, 1979, pp. 145-153.
21. C.L. Jackson and S.L. Tanimoto, "Quadtrees, Oct-tree and K-trees; A generalized approach to recursive decomposition of euclidean space", *IEEE Transactions on PAMI*, Vol. 5(5), 1983, pp. 533-539.
22. L.P. Jones and S.S. Iyenger, "Space and Time efficient virtual quadtrees", *IEEE Transactions on PAMI*, Vol. 6(2), March 1984, pp. 244-247.
23. A. Klinger and R.C. Dyer, "Experiments on picture representation using regular decomposition", *Computer Graphics and Image Processing*, Vol. 5, March 1976, pp. 68-105.
24. A. Klinger and M.L. Rhodes, "Organization and access of image data by areas", *IEEE Transactions on PAMI*, Vol. 1, Jan 1979, pp. 51-60.
25. D.E. Knuth, "The art of Computer programming", Vol. 1, Addison-Wesley publishing company, 2nd edition, 1973.

26. R.G. Loomis, "Boundary networks", *Communications of the ACM*, Vol. 8, Jan 1965, pp. 44-48.

27. R.D. Merril, "Representation of contours and regions for efficient search", *Communications of the ACM*, Vol. 16(2), 1973, pp. 69-81.

28. K. Knowlton, "Progressive transmission of grey scale and B/W pictures by simple, efficient and lossless encoding scheme", *IEEE Proceedings* Vol. 68, 1980, pp. 885-896.

29. W.M. Newman and R.F. Sproull, *Principles of Interactive Computer Graphics*, McGraw Hill Company, 2nd Edition. 1979.

30. M.S. Nortey. "Point in polygon algorithms for geographic information system", M.Sc thesis, University of Alberta, 1982.

31. D.J. Peuquet, "A hybrid structure for the storage and manupulation of very large spatial data sets", *Computer Vision Graphics and Image Processing*, Vol. 24, 1983, pp. 14-27.

32. D.J. Peuquet, "Data structures for spatial data handling", *International Symposium on Spatial Data Handling*, Back ground material to workshop W2, Zurich, Switzerland, August 20-24, 1984.

33. J.L. Pfaltz and A. Rosenfeld, "Computer representation of planar region by their skeletons", *Communications of the ACM*, Vol. 10(2), 1967, pp. 119-122.

34. A. Rosenfeld and J.L. Pfaltz, "Sequential operations in digital picture processing", *Journal of the ACM*, Vol.

13, 1976, pp. 471-496.

35. H. Samet, "Region representation: Quadtrees from binary arrays", *Computer Graphics and Image Processing*, Vol. 13, 1980, pp. 88-93.

36. H. Samet, "Region representation: quadtrees from boundary codes", *Communications of the ACM*, Vol. 23, 1980, pp. 163-170.

37. H. Samet, "An algorithm for converting raster to quadtrees", *IEEE Transactions on PAMI*, Vol. 3(1), 1981, pp. 93-95.

38. H. Samet, "Computing perimeters of regions in images represented by quadtrees", *IEEE Transactions on PAMI*, Vol. 3, 1981, pp. 683-687.

39. H. Samet, "Neighbor finding techniques for images represented by quadtrees", *Computer Graphics and Image Processing*, Vol. 18, 1982, pp. 37-57.

40. H. Samet and R.E. Webber, "Line quadtrees: a hierarchical data structure for encoding of boundaries", *Proceedings Conference on Pattern Recognition and Image Processing, IEEE Computer Society Publication*, 1978, pp 90-92.

41. H. Samet, "The quadtree and related hierarchical data structures", *Computing Surveys*, Vol. 16, No. 2, June 1984, pp. 187-260.

42. M. Shamos and J.L. Bentley, "Optimal algorithms for structuring geographic data", *First International Advanced Study Symposium on Topological Data Structures*

*for Geographic Information Systems*, Vol. 6, 1978, pp S&B/1-S&B/31.

43. M. Shaneier, "Linear time calculations of geometric properties using quadtrees", *Computer Vision, Graphics and Image Processing*, Vol. 16, 1981, pp. 296-302.

44. L.G. Shapiro, "Data structures for picture processing: A survey", *Computer Graphics and Image Processing*, Vol. 11, 1979, pp. 162-184.

45. S.N. Srihari, "Hierarchical data structures and progressive refinement of 3-D images", *IEEE Proceedins on Computer Vision and Pattern Recognition*, 1982, pp. 485-490.

46. M. Tanminen, "Comments on quadtrees and octtrees", *Communications of the ACM*, Vol. 27(3), March 1984, pp. 248-249.

47. S. Tanimoto and T. Pavlidis, "A hierarchical data-structure for picture processing", *Computer Graphics and Image Processing*, Vol. 4, 1975, pp. 104-119.

48. M. Yau and S.N. Srihari, "A hierarchical data structure for multidimensional digital images", *Communications of the ACM*, Vol. 26(7), 1983, pp. 504-515.